Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

|  |
| --- |
|  |
|  |

# оТЧЕТ

по лабораторной работе

на тему:

РЕКУРСИЯ

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил  Студент гр. X5100X |  | В. Н. Протасеня | |
| Проверил |  | Асс. Е.Е. Фадеева | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Минск, 2018

1. Теоретические сведения по теме лабораторной работы

**Рекурсия**– это определение какого-либо понятия через само это понятие.

Существует два вида рекурсии- явная и неявная (взаимная). **Явная рекурсия**– это рекурсия, при которой обращение к подпрограмме содержится в теле самой подпрограммы. **Неявная (взаимная) рекурсия**– это рекурсия, при которой обращение к подпрограмме содержится в теле другой подпрограммы, к которой производится обращение из данной подпрограммы.

Возможность рекурсивного обращения обеспечивается специальными средствами, которые при каждом повторном (рекурсивном) вызове подпрограммы создают новый экземпляр ее памяти, делая временно недоступным прежний экземпляр памяти (т.е. значения локальных переменных и фактических параметров, существующих на момент рекурсивного вызова). Все экземпляры памяти рекурсивной подпрограммы помешаются в стек. Всегда доступен только текущий экземпляр памяти, созданный последним. При каждом возврате из рекурсивной подпрограммы текущий экземпляр памяти удаляется из стека и становится доступным предыдущий экземпляр.

В нерекурсивной подпрограмме механизм выделения памяти в стеке отсутствует.

Рекурсивность является не свойством подпрограммы, а лишь свойством ее описания. Ту же подпрограмму можно организовать и без рекурсии.

**Достоинство рекурсивной записи**– в общем случае запись программы короче и нагляднее, чем нерекурсивная запись.

**Недостатки** – требуется больше памяти и машинного времени (за счет повторных обращений к функции), чем при нерекурсивной записи.

1. Задание на лабораторную работу

Быстрая сортировка

Быстрая сортировка (другое название QuiсkSort) является усовершенствованием метода сортировки обменами (см. подразд. 1.3). Суть метода состоит в следующем:  
**Шаг 1.** Выбирается некоторый элемент ***Х*** массива ***А,*** состоящего из ***n*** элемен- тов, (в качестве такого барьерного элемента может быть выбран центральный элемент массива).  
**Шаг 2.** Массив просматривается в прямом направлении (***i*** = 1, 2... и так далее до центрального элемента) с целью поиска в нем элемента ***A***[***i***], не меньшего, чем ***Х***.  
**Шаг 3.** Массив просматривается в обратном направлении (***j*** = ***n, n-1***,... и так да- лее до центрального элемента), но теперь ведется поиск элемента ***A***[***j***], не пре- восходящего ***Х***.  
**Шаг 4.** Производится перестановка элементов ***A***[***i***] и ***A***[***j***] местами.  
**Шаг 5.** Шаги 1 – 4 повторяются, пока ***i < j.***

В результате таких действий слева от элемента ***Х*** окажутся элементы, меньшие или равные ***Х***, а справа – элементы, большие ***Х.*** Пусть при этом эле- мент ***Х*** попадет в позицию с номером ***k***, тогда массив будет иметь вид

***А*[*1*]*, A*[*2*]*,..., A*[*k-1*]*, A*[*k*]*, A*[*k+1*]*,...,A*[*n*]*.***

Каждый из элементов ***А***[***1***], ***A***[***2***],..., ***A***[***k-1***] меньше или равен ***А***[***k***], а каждый из элементов ***A***[***k+1***],...,***A***[***n***] больше ***А***[***k***]. Отсюда можно сделать вывод, что элемент ***А***[***k***] стоит на своем месте. Исходный массив при этом разделился на две независимые неотсортированные части.

Для дальнейшей сортировки шаги 1–5 применяются для каждой из частей массива. И так до тех пор, пока в массиве не останутся подмассивы, состоящие из одного элемента, т.е. пока массив не будет отсортирован полностью.

1. Выполнение
   1. Разработка алгоритма, используя метод Дамке

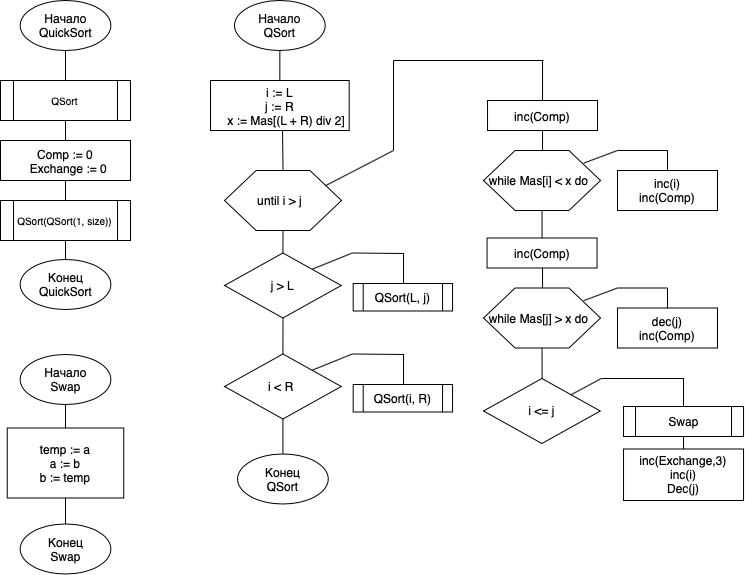


Рисунок 3.1 – Схема работы программы

3.2 Определение подпрограмм и их описание

Таблица 3.1 используемые идентификаторы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя подпрограммы | Назначение подпрограммы | Заголовок подпрограммы | Имя параметра | Назначение параметра |
| **QuickSort** | Внешняя процедура быстрой сортировки | procedure QuickSort(var Mas: Tmas; const size: Integer;var Comp, Exchange: Integer); | Mas | Массив |
| size | Размерность массива |
| Comp | Хранение сравнений |
| Exchange | Хранение замен |
| **QSort** | Внутренняя процедура быстрой сортировки | Procedure QSort(L, R: Integer); | L | Левая граница массива |
| R | Правая граница массива |

* 1. Текст программы и его описание

procedure QuickSort(var Mas: Tmas; const size: Integer; //быстрая сортировка

var Comp, Exchange: Integer);

Procedure QSort(L, R: Integer);

var

i, j, x: Integer;

begin

i := L;

j := R;

x := Mas[(L + R) div 2];

repeat

inc(Comp);

while Mas[i] < x do

begin

inc(i);

inc(Comp);

end;

inc(Comp);

while Mas[j] > x do

begin

Dec(j);

inc(Comp);

end;

if i <= j then

begin

Swap(Mas[i], Mas[j]);

inc(Exchange,3);

inc(i);

Dec(j);

end;

until i > j;

if j > L then

QSort(L, j);

if i < R then

QSort(i, R)

end;

begin

Comp := 0;

Exchange := 0;

QSort(1, size);

end;

* 1. Тестирование и отладка программы

Таблица 3.2 Прохождение тестов программы

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Вводимые данные | Ожидаемый результат | Полученный результат |
| 1. |  |  | Тест пройден |

Программа тесты прошла успешно, ошибок не найдено.

1. Задание на лабораторную работу

Ханойские башни

# Даны три стержня из N дисков разного диаметра, которые надеты на один из 3 стержней (начальный) в порядке убывания диаметра. Надо переместить N дисков за наименьшее число шагов на выбранный(конечный) стержень, так чтобы они остались в таком же порядке. При этом требуется соблюдать правила:

1. На каждом шаге ровно один диск перемещается с одного диска на другой
2. Диск большего диаметра нельзя помещать на диск меньшего диаметра
3. Оставшийся стержень можно использовать как промежуточный
4. Выполнение
   1. Разработка алгоритма, используя метод Дамке

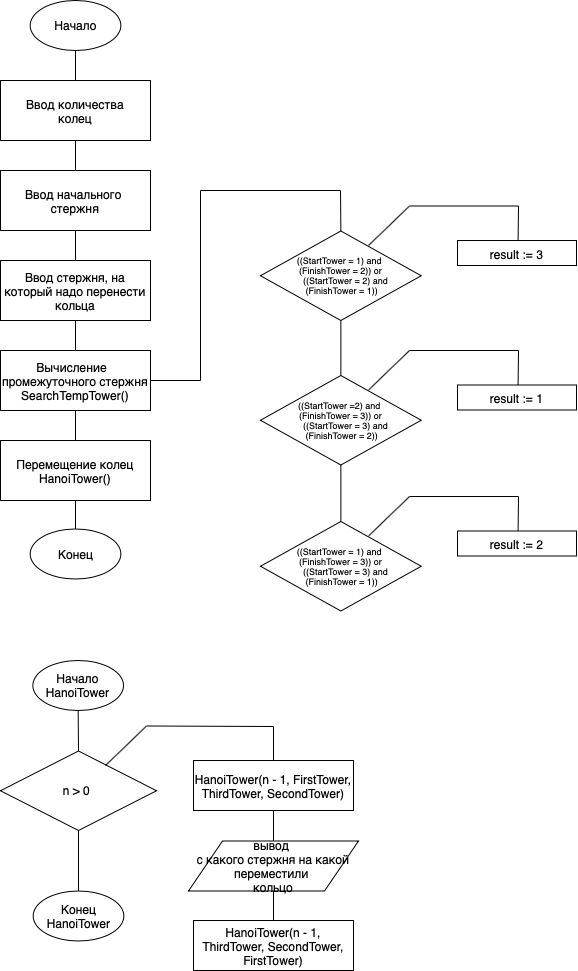


Рисунок 3.1 – Схема работы программы

3.2 Определение подпрограмм и их описание

Таблица 3.1 используемые идентификаторы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя подпрограммы | Назначение подпрограммы | Заголовок подпрограммы | Имя параметра | Назначение параметра |
| **SearchTempTower** | Поиск промежуточкого стержня | function SearchTempTower(StartTower, FinishTower: integer): integer; | StartTower | Начальный стержень |
| FinishTower | Конечный стержень |
| **HanoiTower** | Перемещение колец с одного стержня на другой | procedure HanoiTower(n: integer; s1, s2, s3: integer); | n | Количество дисков |
| S1 | Стержень 1 |
| S2 | Стержень 2 |
| S3 | Стержень 3 |

* 1. Текст программы и его описание

program laba3\_part2;

APPTYPE CONSOLE}

{$R \*.res}

uses

System.SysUtils;

var

RingsCount, StartTower, FinishTower, TempTower: integer;

function SearchTempTower(StartTower, FinishTower: integer): integer; //поиск промежуточного стержня

begin

if ((StartTower = 1) and (FinishTower = 2)) or

((StartTower = 2) and (FinishTower = 1)) then

result := 3;

if ((StartTower = 2) and (FinishTower = 3)) or

((StartTower = 3) and (FinishTower = 2)) then

result := 1;

if ((StartTower = 1) and (FinishTower = 3)) or

((StartTower = 3) and (FinishTower = 1)) then

result := 2

end;

procedure HanoiTower(n: integer; s1{1-стержень}, s2{2-1 стержень}, s3{3-й стержень}: integer);// рекурсивная процедура

//перемещения колец

begin

if n > 0 then

begin

HanoiTower(n - 1, s1, s3, s2);

Writeln(s1, ' -> ', s3); //вывод сообщение о перемещении 1 кольуа

HanoiTower(n - 1, s3, s2, s1);

end

end;

begin

Writeln('Введите количество колец:');

readln(RingsCount);

if RingsCount = 0 then

Writeln('Вы ввели некорректное значение. Попробуйте еще раз!')

else

begin

Writeln('Введите начальный стержень (значение от 1 до 3):');

readln(StartTower);

Writeln('Введите стержень, на который необходимо переместить кольца:');

readln(FinishTower);

TempTower := SearchTempTower(StartTower, FinishTower); // поиск промежуточного

if (FinishTower > 3) or (StartTower > 3) or (FinishTower = StartTower) then

Writeln('Вы ввели некорректное значение. Попробуйте еще раз!')

else

begin

Writeln('Результат перемещений колец с одного стержня на другой:');

HanoiTower(RingsCount, StartTower, FinishTower, TempTower); //рекурсивная процеду перемещения колец

end;

end;

readln;

end.

* 1. Тестирование и отладка программы

Таблица 3.2 Прохождение тестов программы

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Вводимые данные | Ожидаемый результат | Полученный результат |
| 1. |  |  | Тест пройден |
| 2. |  |  | Тест пройден |
| 3. |  |  | Тест пройден |
| 4. |  |  | Тест пройден |

Программа тесты прошла успешно, ошибок не найдено.

1. Задание на лабораторную работу

Перестановки элементов в массиве

Дан массив, состоящий из N элементов. Необходимо найти и вывести все перестановки элементов в массиве. Число перестановок= N!

1. Выполнение
   1. Разработка алгоритма, используя метод Дамке

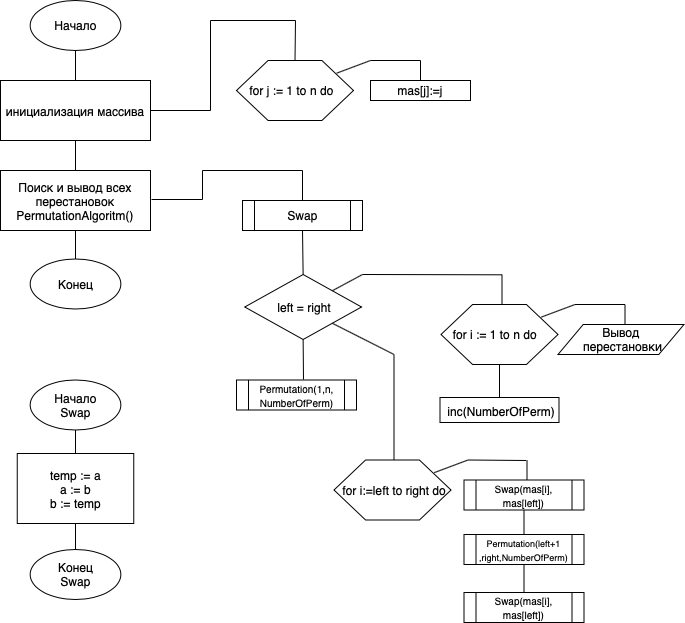


Рисунок 3.1 – Схема работы программы

3.2 Определение подпрограмм и их описание

Таблица 3.1 используемые идентификаторы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя подпрограммы | Назначение подпрограммы | Заголовок подпрограммы | Имя параметра | Назначение параметра |
| **PermutationAlgoritm** | Внешняя процедура для перестановок | procedure PermutationAlgoritm(left, right:integer; var NumberOfPerm: integer; var mas:Tmas); | left | Левая граница массива |
| right | Правая граница массива |
| NumberOfPerm | Количество перестановок |
| mas | Массив с элементами для перестановки |
| **Permutation** | Внутренняя процедура для перестановок | procedure Permutation(left, right: integer; var NumberOfPerm: integer ); | left | Левая граница массива |
| right | Правая граница массива |
| NumberOfPerm | Количество перестановок |
| **Swap** | Обмен элементов в массиве | procedure Swap(var a, b: integer); | a,b | Элементы массива |

* 1. Текст программы и его описание

program laba3\_part3;

{$APPTYPE CONSOLE}

{$R \*.res}

uses

System.SysUtils;

const

n = 4; // количество элементов в массиве

type

TMas= array [1 .. n] of integer;

var

j: integer;

NumberOfPerm: integer;// количество перестановок

Mas: Tmas;

procedure PermutationAlgoritm(left, right:integer; var NumberOfPerm: integer; var mas:Tmas);

procedure Permutation(left, right: integer; var NumberOfPerm: integer );

var

i: integer;

procedure Swap(var a, b: integer);

var

temp: integer;

begin

temp := a;

a := b;

b := temp;

end;

begin

if left = right then

begin

for i := 1 to n do // вывод перестановки

write(mas[i]);

writeln;

inc(NumberOfPerm); //подсчет количества перестановок

end

else

for i:=left to right do //сдвиг левой границы

begin

Swap(mas[i], mas[left]);

Permutation(left+1,right,NumberOfPerm);

Swap(mas[i], mas[left])

end

end;

begin

Permutation(1,n,NumberOfPerm);

end;

begin

writeln('Все перестановки элементов в массиве:');

for j := 1 to n do

mas[j]:=j;

PermutationAlgoritm(1,n,NumberOfPerm,mas);

writeln;

writeln('Количество перестановок: ',NumberOfPerm);

readln;

readln;

end.

* 1. Тестирование и отладка программы

Таблица 3.2 Прохождение тестов программы

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Вводимые данные | Ожидаемый результат | Полученный результат |
| 1. | N=3 |  | Тест пройден |
| 2. | N=4 |  | Тест пройден |

Программа тесты прошла успешно, ошибок не найдено.