*Федеральное государственное бюджетное образовательное учреждение высшего образования*

***«Московский государственный технический университет имени Н.Э. Баумана»***

***(национальный исследовательский университет)***

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

# ОТЧЕТ
# ПО ЛАБОРАТОРНОЙ РАБОТЕ №1
# Вариант №7

**Дисциплина:**

Технологии разработки программных систем

**Название лабораторной работы:**

Исследование структур и методов обработки данных

Студент гр. ИУ6-42 _____ Р.Д. Векшин

подпись, дата И.О. Фамилия

Преподаватель _____ К.С. Хорунжина

подпись, дата И.О. Фамилия

Москва 2018

# СОДЕРЖАНИЕ

ВВЕДЕНИЕ....4
1 Цель работы....4
2 Задача....4
3 Исходные данные....5
ОСНОВНАЯ ЧАСТЬ....6
1 Исходные варианты структуры и методов её обработки....6
Листинг 1 — Структуры данных и записи....6
1.1 Определение объёма данных....6
1.2 Анализ алгоритма поиска....8
Листинг 2 — Двоичный поиск индекса элемента в статическом массиве....8
Оценка времени поиска i-го элемента структуры....8
1.3 Анализ алгоритма упорядочения....9
Листинг 3 — Упорядочение статического массива методом вставки....9
Оценка времени упорядочения структуры методом вставки....9
1.4 Анализ алгоритма удаления....10
Листинг 4 — Удаление сдвигом элемента статического массива....10
Оценка времени удаления i-го элемента структуры....10
1.5 Вывод....10
2 Альтернативные варианты структуры и методов её обработки....11
Листинг 5 — Структуры данных и записи....11
2.1 Определение объёма данных....11
2.2 Анализ алгоритма поиска....12
Листинг 6 — Двоичный поиск индекса элемента в динамическом массиве....12
Оценка времени поиска i-го элемента структуры....12
2.3 Анализ алгоритма упорядочения....13
Листинг 7 — Упорядочение динамического массива по алгоритму Шелла....13
Оценка времени упорядочения структуры по алгоритму Шелла....13
2.4 Анализ алгоритма удаления....14
Листинг 8 — Удаление сдвигом элемента динамического массива....14
Оценка времени удаления i-го элемента структуры....14
2.5 Вывод....14
ЗАКЛЮЧЕНИЕ....15

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....16
Литература.....16
ПРИЛОЖЕНИЕ А.....17
Листинг операторов сравнения для записи Component.....17

# ВВЕДЕНИЕ

При разработке алгоритмов программ часто возникает задача выбора структур данных и методов их обработки. Исходными составляющими для решения этой задачи являются описание набора и типов хранимых данных, а также перечень операций, выполняемых над ними.

Можно выделить следующие основные вопросы, на которые необходимо ответить при решении поставленной задачи:

а) Как логически организовать структуру данных? Как ее реализовать?

б) Как осуществлять поиск информации? Как упорядочить данные?

в) Как выполнить функции корректировки данных?

## 1 Цель работы

Исследование структуры структур данных,методов их обработки и оценки.

## 2 Задача

а) Ознакомиться с теоретическими сведениями по абстрактным структурам данных и методами их обработки.

б) Для указанной задачи и типа данных предложить способ реализации и определить требуемый объем памяти.

в) Провести анализ заданных методов поиска, упорядочения и корректировки. Оценить время выполнения соответствующих операций.

г) Предложить альтернативный вариант решения задачи, в котором должно быть минимум одно улучшение. Улучшения могут касаться как структуры данных, так и основных операций. Обосновать новые решения, используя количественные и качественные критерии. Количественными критериями являются: объем памяти, среднее количество сравнений и количество тактов. Качественные критерии определяют возможность использования того или иного метода применительно к разработанной структуре. К ним можно отнести: применимость операции только к упорядоченным данным; необходимость знать количество элементов; наличие признака разбивки на гнезда; необходимость в прямом доступе к элементам; знание граничных значений; невозможность создать структуру в соответствии с арифметической прогрессией и др.

# 3 Исходные данные

Задача 2: Дана таблица материальных нормативов, состоящая из K записей фиксированной длины вида: код детали; код материала; единица измерения; номер цеха; норма расхода.

Структура: Таблица;

Поиск: Метод дихотомии (двоичный поиск);

Упорядочение: Вставка;

Корректировка: Удаление сдвигом.

# ОСНОВНАЯ ЧАСТЬ

## 1 Исходные варианты структуры и методов её обработки

Для хранения записи Component использован статический массив Components размера K и индексами от 0 до K-1 (см. рис. 1).

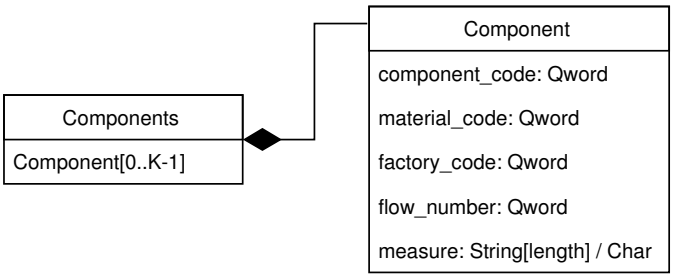


*Рисунок 1 — Диаграмма статического массива и его элементов*

В листинге 1 представлены реализации структуры Components и записи Component на языке Pascal.

***Листинг 1 — Структуры данных и записи***

```
type Component = record
  component_code: QWord;        // один из Byte/Word/LongWord/QWord
  material_code:  QWord;        // один из Byte/Word/LongWord/QWord
  factory_code:   QWord;        // один из Byte/Word/LongWord/QWord
  flow_number:    QWord;        // один из Byte/Word/LongWord/QWord
  measure:        String[255];  // один из string[length]/char
end;

type
  Components = Array [0 .. K-1] of Component;
```

### 1.1 Определение объёма данных

Минимальный объем данных, занимаемой одной записью:

$$V_{ST2} = V(Byte) * 4 + V(Char) = 1 * 4 + 1 = 5 \text{ Байт.}$$

Максимальный объем памяти, занимаемой массивом, содержащим K записей:

$$V_{arr2} = K * V_{ST2} = K * 5 \text{ Байт.}$$

Максимальный объем памяти, занимаемой одной записью:

$$V_{ST1} = V(QWord) * 4 + V(String[255]) = 8 * 4 + 256 = 288 \text{ Байт.}$$

Максимальный объем памяти, занимаемой массивом, содержащим K записей:

$$V_{arк1} = K * V_{ST1} = K * 288 \text{ Байт.}$$

Заметим, что если диапазон значений кода детали, кода материала, номера цеха; нормы расхода находится: в пределах от 0 до $2^8$-1 включительно, то данные поля записи можно определить типом «Byte», который займет в памяти 1 байт на каждое поле; в пределах от 0 до $2^{16}$-1 включительно, то данные поля записи можно определить типом «Word», который займет в памяти 2 байта на каждое поле; в пределах от 0 до $2^{32}$-1 включительно, то данные поля записи можно определить типом «LongWord», который займет в памяти 4 байта на каждое поле; в пределах от 0 до $2^{64}$-1 включительно, то данные поля записи можно определить типом «QWord», который займет в памяти 8 байта на каждое поле.

Поле measure (единица измерения), если его значение записывается одним символом, можно определить типом «Char», который займет в памяти 1 байт, если диапазон его значений находится в интервале от 2 до 255 символов, то данное поле записи можно определить типом «String[length]», где length — длина, который занимает в памяти 1 + length Байт.

В любом из возможных случаев определения типа полей записи размер массива линейно зависит от размера его элементов и определяется по формуле $V_{arr} = K * V_{ST}$, где $V_{ST}$ — объем памяти, занимаемой одной записью, K — целочисленная константа, определяющая размер массива записей. Заметим, что объем памяти, который будет выделен операционной системой для хранения переменной типа Components, является постоянным на всём протяжении времени жизни данной переменной и не зависит от количества инициализированных переменных типа Component рассматриваемого массива.

При расчете объемов памяти, занимаемых записью и массивом, не учитывалось выравнивание данных в оперативной памяти.

## 1.2 Анализ алгоритма поиска

По заданию необходимо реализовать и оценить способ поиска методом дихотомии. Двоичный (бинарный) поиск (или дихотомия) — классический алгоритм поиска элемента в отсортированном массива, использующий дробление массива на половины.

В листинге 2 представлена реализация алгоритма двоичного поиска в структуре Components на языке Pascal.

***Листинг 2 — Двоичный поиск индекса элемента в статическом массиве***

```
// Двоичный поиск по сортированному массиву
function find ( arr: Components; elem: Component; size: QWord ): Int64;
var m, i, j: QWord;
begin
  i := 0;
  j := size - 1;
  m := (i + j) >> 1

  while (arr[m] <> elem) and (i <= j) do
  begin
    if elem > arr[m] then
      i := m + 1
    else
      j := m - 1;
    m := (i + j) >> 1;
  end;

  if i > j then
    find := -1
  else
    find := m;
end;
```

***Оценка времени поиска i-го элемента структуры***

$$T = t_{установки\,нач.данных} + t_{цикла} + t_{выхода} = (3t_{:=} + t_{a+const} + t_{a+b} + t_{>>}) +$$

$$+ [\log_{2N}]((t_{a[i]} + t_{<>} + t_{<=}) + (t_{a[i]} + t_{>} + t_{:=} + t_{a+const}) + (t_{:=} + t_{a+b} + t_{>>})) + t_{a[i]} + t_{<>} + t_{>} + t_{:=} =$$

$$= (2+3+4) + ([\log_2 N] * (4+3+3+2) + 3) + (1+2) = 15 + 12[\log_2 N]$$

## 1.3 Анализ алгоритма упорядочения

По заданию необходимо реализовать и оценить упорядочение массива методом вставки.

В листинге 3 представлена реализация алгоритма упорядочения записей структуры Components на языке Pascal.

***Листинг 3 — Упорядочение статического массива методом вставки***

```
// Сортировка вставками
procedure sort ( var arr: Components; size: QWord );
var
  i, j: QWord;
  t:    Component;
begin
  for i := 1 to size - 1 do
    begin
      j := i;
      while (j > 0) and (arr[j] < arr[j - 1]) do
      begin
        // count += 1
        t := arr[j];
        arr[j] := arr[j - 1];
        arr[j - 1] := t;
        j -= 1;
      end
    end;
end;
```

***Оценка времени упорядочения структуры методом вставки***

| Размер массива | Количество перестановок |
|---:|---:|
| 2 | 1 |
| 4 | 6 |
| 8 | 28 |
| 16 | 120 |
| 32 | 496 |
| 64 | 2 016 |
| 128 | 8 128 |
| 256 | 32 640 |

## 1.4 Анализ алгоритма удаления

По заданию необходимо реализовать и оценить удаление сдвигом.

В листинге 4 представлена реализация алгоритма удаления записей из структуры Components на языке Pascal.

***Листинг 4 — Удаление сдвигом элемента статического массива***

```
// Процедура удаления записи из массива
procedure del ( var arr: Components; index: QWord; var size: QWord );
var
  i: QWord;
begin
  for i := index to size - 2 do
    arr[i] := arr[i+1];
  size -= 1;
  arr[size] := nil;
end;
```

***Оценка времени удаления i-го элемента структуры***

$$T = t_{установки\,нач.\,индекса} + t_{проверки\,условия} + 1 + n\left(t_{тела\,цикла} + 2 + 2\right) = 5 + n\left(t_{тела\,цикла} + 4\right) =$$

$$= 5 + n\left(2t_{a[i]} + 3t_{:=} + 2t_{a+const} + 4\right) = 5 + n\left(2*2 + 3*2 + 2*1 + 4\right) = 5 + 16n$$

## 1.5 Вывод

Использование статического массива оправдано для небольших объемов данных. Однако для реализации возможности удаления элементов необходимо хранить в памяти переменную, содержащую используемый размер массива.

Сортировка вставками обеспечивает упорядочение структуры за квадратичное время.

# 2 Альтернативные варианты структуры и методов её обработки

Для хранения записи Component использован динамический массив Components (см. Рис. 2).

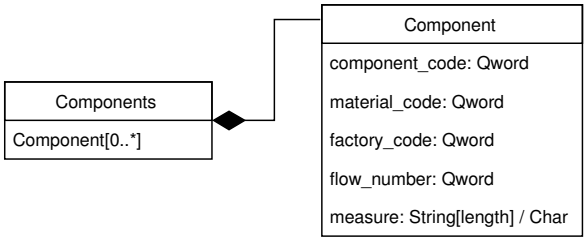


*Рисунок 2 — Диаграмма динамического массива и его элементов*

В листинге 5 представлены реализации структуры Components и записи Component на языке Pascal.

***Листинг 5 — Структуры данных и записи***

```
type Component = record
  component_code: QWord;        // один из Byte/Word/LongWord/QWord
  material_code:  QWord;        // один из Byte/Word/LongWord/QWord
  factory_code:   QWord;        // один из Byte/Word/LongWord/QWord
  flow_number:    QWord;        // один из Byte/Word/LongWord/QWord
  measure:        String[255];  // один из string[length]/char
end;

type
  Components = Array of Component;
```

Здесь и далее полужирным стилем выделены изменённые или добавленный части программного кода.

## 2.1 Определение объёма данных

Минимальный объем данных, занимаемой одной записью:

$$V_{ST2} = V(Byte) * 4 + V(Char) = 1 * 4 + 1 = 5 \text{ Байт.}$$

Максимальный объем памяти, занимаемой массивом, содержащим N записей:

$$V_{arr2} = N * V_{ST2} = N * 5 \text{ Байт.}$$

Максимальный объем памяти, занимаемой одной записью:

$$V_{ST1} = V(QWord) * 4 + V(String[255]) = 8 * 4 + 256 = 288 \text{ Байт.}$$

Максимальный объем памяти, занимаемой массивом, содержащим N записей:

$$V_{arк1} = N * V_{ST1} = N * 288 \text{ Байт.}$$

Размер памяти, занимаемой одной записью, определяется по тем же правилам, что и в исходном варианте структуры.

Размер массива линейно зависит от размера его элементов и определяется по формуле $V_{arr} = N * V_{ST}$, где $V_{ST}$ — объем памяти, занимаемой одной записью, N — целочисленная переменная, определяющая размер массива записей и способная изменяться в течение времени работы программы. Данный вариант описания структуры данных полезен для таких случаев, когда разработчик не знает на момент компиляции программного кода точное количество элементов массива, которые будут определены пользователем.

## 2.2 Анализ алгоритма поиска

В листинге 6 представлена реализация алгоритма двоичного поиска записи в структуре Components на языке Pascal.

***Листинг 6 — Двоичный поиск индекса элемента в динамическом массиве***

```
// Двоичный поиск по сортированному массиву
function find ( arr: Components; elem: Component ): Int64;
var m, i, j: QWord;
begin
  i := low(arr);
  j := high(arr);
  m := (i + j) >> 1

  while (arr[m] <> elem) and (i <= j) do
  begin
    if elem > arr[m] then
      i := m + 1
    else
      j := m - 1;
    m := (i + j) >> 1;
  end;

  if i > j then
    find := -1
  else
    find := m;
end;
```

***Оценка времени поиска i-го элемента структуры***

$$T = t_{установки\,нач.данных} + t_{цикла} + t_{выхода} = (2 + 3t_{:=} + t_{a+const} + t_{a+b} + t_{>>}) \; +$$

$$+ \; [\log_2 N]((t_{a[i]} + t_{<>} + t_{<=}) + (t_{a[i]} + t_{>} + t_{:=} + t_{a+const}) + (t_{:=} + t_{a+b} + t_{>>})) + t_{a[i]} + t_{<>} + t_{>} + t_{:=} \; =$$

$$= \; (2+2+3+4) + ([\log_2 N] * (4+3+3+2) + 3) + (1+2) = 17 + 12[\log_2 N]$$

## 2.3 Анализ алгоритма упорядочения

Реализуем и оценим метод упорядочения по алгоритму Шелла.

В листинге 7 представлена реализация алгоритма упорядочения элементов структуры Components на языке Pascal.

***Листинг 7 — Упорядочение динамического массива по алгоритму Шелла***

```
...
Uses math;
...
// Сортировка Шелла
procedure shell_sort ( var arr : Components );
var
  i, j, k, step, n, powr: QWord;
  tmp : Component;
begin
  n := length(arr);
  powr := trunc(log2(length(arr) + 1));
  for k := powr downto 1 do
  begin
    step := round((2 << (k-1)) - 1);
    for i := step to n do
    begin
      tmp := arr[i];
      j := i;
      while (j >= step) and (tmp < arr[j - step]) do
      begin
        // count += 1;
        arr[j] := arr[j - step];
        j := j - step;
      end;
      // count += 1;
      arr[j] := tmp;
    end;
  end;

end;
```

***Оценка времени упорядочения структуры по алгоритму Шелла***

| Размер массива | Количество перестановок |
|---:|---:|
| 2 | 5 |
| 4 | 10 |
| 8 | 24 |
| 16 | 70 |
| 32 | 168 |
| 64 | 442 |
| 128 | 1 020 |
| 256 | 2 494 |

## 2.4 Анализ алгоритма удаления

В листинге 8 представлена реализация алгоритма удаления записей из структуры Components на языке Pascal.

***Листинг 8 — Удаление сдвигом элемента динамического массива***

```
// Процедура удаления записи из массива
procedure del ( var arr: Components; index: QWord );
var
  i: QWord;
begin
  for i := index to high(arr) - 1 do
    arr[i] := arr[i+1];
  setlength(arr, length(arr) - 1);
end;
```

***Оценка времени удаления i-го элемента структуры***

$$T = t_{установки\,нач.\,индекса} + t_{проверки\,условия} + 1 + n\left(t_{тела\,цикла} + 2 + 2\right) = 6 + n\left(t_{тела\,цикла} + 4\right) =$$

$$= 5 + n\left(2t_{a[i]} + 2t_{:=} + 2t_{a+const} + t_{getlength(arr)} + 4\right) = 6 + n\left(2*2 + 2*2 + 2*1 + 2 + 4\right) = 6 + 16n$$

## 2.5 Вывод

Использование динамических массивов позволяет добиться более эффективного использования памяти, выделяемой под структуру, по сравнению со статическими массивами.

Сортировка Шелла в худшем случае обеспечивает упорядочение массива за $O(n^2)$, но с меньшим коэффициентом амортизации по сравнению с сортировкой вставками.

# ЗАКЛЮЧЕНИЕ

*Таблица 1 — Сравнительная таблица исходного и альтернативного вариантов*

| | **Исходный вариант** | **Альтернативный вариант** |
|---|---|---|
| **Структура данных** | Статический массив (статическая таблица) | Динамический массив (динамическая таблица) |
| **Объем памяти** | K*V(элемент),<br>K = CONST | N*V(записи),<br>N — число введенных записей |
| **Алгоритм поиска** | Дихотомия | Дихотомия |
| **Сложность поиска** | $15 + 12[\log_2 N]$ | $17 + 12[\log_2 N]$ |
| **Алгоритм упорядочения** | Метод вставки | Алгоритм Шелла |
| **Объем данных** | Количество перестановок | Количество перестановок |
| **2** | 1 | 5 |
| **4** | 6 | 10 |
| **8** | 28 | 24 |
| **16** | 120 | 70 |
| **32** | 496 | 168 |
| **64** | 2 016 | 442 |
| **128** | 8 128 | 1 020 |
| **256** | 32 640 | 2 494 |
| **Корректировка** | Удаление 1 элемента по индексу со сдвигом оставшейся части | Удаление 1 элемента по индексу со сдвигом оставшейся части |
| **Сложность корректировки** | 5 + 16n | 6 + 16n |

Увеличение количества тактов в методах поиска и корректировки для альтернативного варианта структуры происходит из-за операций *low(arr)*, *high(arr)*, *length(arr)*, но при этом в данные методы не нужно передавать информацию о размере массива и использовать *магические числа* для указания начальных и конечных индексов массива. Использование динамических массивов позволяет более эффективно использовать память для хранения структуры, т. к. нет необходимости хранить в отдельной переменной используемый размер для статического массива и вся аллоцированная память заполняется полезными данными (элементами структуры). В худшем случае сортировка Шелла, как и сортировка вставками, обеспечивает упорядочение структуры за квадратичное время, но, в отличие от сортировки вставками, сортировка происходит с меньшим коэффициентом амортизации.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

## Литература

**Иванова Г.С., Пугачев Е.К.** Исследование структур и методов обработки данных. Методические указания по выполнению лабораторной работы по дисциплине «Технология программирования» — Москва, 2013. -15с.

# ПРИЛОЖЕНИЕ А

***Листинг операторов сравнения для записи Component***

```
// Оператор сравнения “=” по коду детали
operator < ( a, b: Component ) R: Boolean;
begin
  R := a.component_code < b.component_code;
end;

// Оператор сравнения “>” по коду детали
operator > ( a, b: Component ) R: Boolean;
begin
  R := a.component_code > b.component_code;
end;

// Оператор сравнения “<” по коду детали
operator = ( a, b: Component ) R: Boolean;
begin
  R := a.component_code = b.component_code;
end;
```