МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное автономное образовательное учреждение высшего образования
«Севастопольский государственный университет»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к лабораторным работам по дисциплине “ОПЕРАЦИОННЫЕ СИСТЕМЫ”, часть 1,

для студентов ОФО и ЗФО направлений подготовки:
09.03.01 «Информатика и вычислительная техника»
09.03.02 «Информационные системы и технологии»
27.03.04 «Управление в технических системах»

Севастополь
2019

УДК 004.451

Методические указания к выполнению лабораторных работ по дисциплине «Операционные системы», часть 1, для студентов ОФО и ЗФО направлений подготовки: 09.03.01 «Информатика и вычислительная техника», 09.03.02 «Информационные системы и технологии», 27.03.04 «Управление в технических системах» / Сост. ст. преподаватель кафедры ИТиКС Е.М. Шалимова. – Севастополь: Изд-во СевГУ, 2019. – 16с.

Целью методических указаний является оказание помощи студентам в выполнении лабораторных работ по дисциплине «Операционные системы».

Приведены теоретические сведения, необходимые для выполнения лабораторных работ, варианты заданий, рекомендации по выполнению, требования к отчетам.

Методические указания рассмотрены и утверждены на заседании кафедры «Информационные технологии и компьютерные системы», протокол № 1 от 31.08. 2018г.

.

Рецензент: доцент кафедры «Информационные технологии и компьютерные системы», к.т.н. Е.В. Козлова.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В соответствии с программой дисциплины «Операционные системы» студент должен выполнить семь лабораторных работ.

По результатам выполнения каждой лабораторной работы оформляется и защищается отчет.

По результатам выполнения каждой лабораторной работы оформляется и защищается отчет, который должен содержать следующие основные элементы:

- титульный лист;
- цель работы;
- постановку задачи;
- описание алгоритма;
- описание используемых структур данных;
- спецификации подпрограмм;
- тестовые примеры;
- текст программы;
- результаты и выводы.

Отчёты по лабораторной работе оформляются и защищаются каждым студентом индивидуально. При защите лабораторной работы студент должен продемонстрировать работу программы.

Разработанные программы должны обеспечивать ввод исходных данных, проверку их корректности, вывод исходных данных и результатов. Логически законченные фрагменты должны быть оформлены в виде подпрограмм, все необходимые данные которым передаются через список параметров. Глобальные переменные не использовать.

1. ЛАБОРАТОРНАЯ РАБОТА №1. РАЗРАБОТКА, ОТЛАДКА И ТЕСТИРОВАНИЕ ПРОГРАММ, ВЫПОЛНЯЕМЫХ ПОД УПРАВЛЕНИЕМ ОС WINDOWS

(входной контроль)

Цель работы: изучение возможностей, предоставляемых ОС Windows для разработки, отладки и тестирования программ**.**

1.1.Задания на лабораторную работу

***Задание 1*:**

1) разработать алгоритм и программу на языке Pascal/C, в которой ввести одномерный массив x, содержащий n элементов заданного типа, выполнить обработку в соответствии с вариантом (таблица 1). исходный массив и результаты вывести на экран;
2) разработать тестовые примеры;
3) отладить программу;
4) проанализировать полученные результаты;
5) оформить отчет.

Таблица 1-Варианты индивидуальных заданий

№варианта	Формулировка задания
1	Найти сумму отрицательных и произведение положительных элементов массива.
2	Переставить минимальный и максимальный элементы массива.
3	Упорядочить элементы массива по возрастанию.
4	Упорядочить элементы массива по убыванию.
5	Найти среднее арифметическое всех положительных элементов массива.
6	Определить количество положительных и количество отрицательных элементов массива, абсолютная величина которых не превосходит X.
7	Преобразовать массив, расположив в нем элементы в обратной последовательности.
8	Преобразовать массив, заменив все отрицательные числа на -1,а положительные на +1.
9	Поменять местами первый и максимальный элементы массива.
10	Найти минимальный и максимальный элементы массива и их номера.
11	Найти среднее арифметическое всех отрицательных элементов массива, расположенных после первого нулевого элемента.
12	Определить количество положительных, количество отрицательных и количество нулевых элементов массива.
13	“Сжать” массив, удалив из него все нулевые элементы.
14	Преобразовать массив, поменяв первый элемент с последним, второй с предпоследним и т.д.
15	Преобразовать массив, расположив сначала все положительные, а затем все отрицательные элементы. Порядок следования элементов одного типа не менять.
16	Найти произведение всех положительных элементов массива, расположенных после первого нулевого элемента.
17	Поменять местами минимальный элемент массива и первый положительный элемент.
18	Определить количество элементов массива кратных 5, значения которых не превосходят X**.**

19	Найти среднее арифметическое элементов массива, лежащих в диапазоне от A до B.
20	Определить максимальный по абсолютной величине элемент массива и переставить его с последним нулевым элементом массива.

Задание 2:

1) разработать алгоритм и программу на языке Pascal/C, в которой ввести двумерный массив X, содержащий N строк и M столбцов элементов заданного типа; выполнить обработку по варианту (таблица 2). Исходный массив и результаты вывести на экран, при этом двумерный массив выводить в форме матрицы;
2) разработать тестовые примеры;
3) отладить программу;
4) проанализировать полученные результаты;
5) оформить отчет.

Таблица 2-Варианты индивидуальных заданий

№варианта	Формулировка задания
1	Преобразовать массив: элементы строки, в которой находится максимальный элемент матрицы, заменить нулями.
2	Преобразовать массив: элементы того столбца, в котором находится максимальный элемент матрицы, заменить нулями.
3	Каждый столбец массива упорядочить по убыванию.
4	Сформировать одномерный массив, состоящий из максимальных элементов каждого столбца.
5	Преобразовать массив: разделить элементы каждого столбца заданной матрицы на последний элемент столбца.
6	Преобразовать массив: разделить элементы каждой строки матрицы на последний элемент этой строки.
7	Определить количество нулевых элементов в каждой строке матрицы.
8	Определить количество нулевых элементов в каждом столбце матрицы.
9	Сформировать одномерный массив, состоящий из сумм положительных элементов каждого столбца.
10	Сформировать одномерный массив, состоящий из сумм отрицательных элементов каждой строки.
11	Найти среднее арифметическое положительных элементов каждой строки.
12	Найти среднее арифметическое положительных элементов каждого столбца.
13	Преобразовать массив, умножив элементы каждой строки на минимальный элемент этой строки.
14	Преобразовать массив, умножив элементы каждого столбца на минимальный элемент этого столбца.
15	Найти количество нулевых элементов в каждой строке матрицы.
16	Найти количество нулевых элементов в каждом столбце матрицы.
17	Преобразовать массив: разделить элементы каждой строки матрицы на абсолютную величину первого элемента этой строки.
18	Преобразовать массив: разделить элементы каждой строки матрицы на среднее арифметическое элементов этой строки.
19	Преобразовать массив: умножить элементы каждого столбца матрицы на среднее арифметическое элементов этого столбца.
20	Сформировать одномерный массив, состоящий из средних арифметических положительных элементов каждой строки.

2. ЛАБОРАТОРНАЯ РАБОТА №2. ВВОД-ВЫВОД И ОБРАБОТКА СТРОК В ЯЗЫКЕ С

Цель работы: программирование ввода-вывода в языке С, изучение функций обработки строк.

2.1. Основные теоретические положения

Функции ввода-вывода объявлены в заголовочном файле ***<stdio.h>.***

При запуске С-программы операционная система всегда открывает три стандартных файла (или потока): входной (файловый указатель ***stdin***), выходной (stdout) и файл ошибок (***stderr)***. Обычно ***stdin*** соотнесен с клавиатурой, ***stdout*** и ***stderr*** –с экраном.

Любой другой файл предварительно должен быть открыт. Для этого используется функция ***fopen***:

FILE* fopen(char* name, char* mode);

Функция получает в качестве аргументов внешнее имя файла (***name)*** и режим доступа ***(mode),*** а возвращает файловый указатель, используемый в дальнейшем для работы с файлом. Функция возвращает нулевой указатель, если файл не может быть открыт по каким либо причинам.

Файловый указатель ссылается на специальную структуру типа ***FILE,*** содержащую информацию о файле.

Режим доступа может принимать значения, указанные в таблице 2.1.

Таблица 2.1- Режимы доступа к файлу

"r"	- файл открывается только для чтения;
"w"	- файл создается только для записи, при этом, если он уже существовал, его содержимое теряется;
"a"	- файл создается или открывается для записи в конец файла;
"r+"	- файл открывается для чтения и для записи;
"w+"	- файл создается для чтения и для записи, при этом, если он уже существовал, его содержимое теряется;
"a+"	- файл создается или открывается для чтения и для записи в конец файла

После окончания работы файл должен быть закрыт с помощью функции ***fclose:.***

int fclose(FILE*);

Если программа завершается нормально, все открытые файлы автоматически закрываются системой.

При работе с файлами существуют различия, определяемые набором функций, используемых для ввода/вывода данных. В частности, рассмотрим форматированный ввод, для этого используется функция

int fscanf(FILE *f,const char *format [, p1,p2,. . .]);

Функция читает текстовые данные из файла ***f*** , преобразует их в соответствии со спецификациями, содержащимися в формате и присваивает по порядку аргументам ***p1,p2,…,*** каждый из которых должен быть указателем.

Для чтения из стандартного файла используется функция

int scanf(const char *format [, p1,p2,. . .]);

Обратные действия производит функция форматированного вывода

int fprintf(FILE *f,const char *format, . . .) ;

Функция преобразует аргументы (список которых обозначен многоточием) в текстовое представление в соответствии с форматом (***format)*** и пишет в выходной файл ***f***. Для вывода в стандартный файл используется функция

int printf(const char *format, . . .) ;

Спецификации форматов для функций семейств ***scan*** и ***print*** даны в приложении А.

Пример1. Рассмотрим пример обработки строки символов. Для заданной строки вывести все символы, расположенные между первой и второй запятыми. Считать, что слова разделены пробелами; строка заканчивается точкой и не может содержать больше 20 символов.

При решении данной задачи необходимо:

1. Определить адрес первой ’,’.
2. Определить адрес второй ’,’.
3. Если в строке нет двух ’,’ , напечатать соответствующее сообщение, иначе определить длину искомой подстроки и напечатать ее.

Текст программы:

```
#include <stdio.h>
#include <string.h>

 main()
{ int i=0,j;
 char x[20];  char*u1,*u2;

   scanf("%c",&x[0]);
   while (x[i]!='.')
   { i++;
 scanf("%c",&x[i]); }
u1= strchr(x,',') ; *u1=' ' ; u2= strchr(x,',') ;
if((u1==NULL) ||(u2==NULL) )
    printf("The string hasn't got two ,") ;
else {
      i=u2-u1;
    for(j=0;j<i;j++)
   printf("%c",u1[j]);
       }
}
```

Рассмотрим работу с файлами, содержащими текстовую информацию, разбитую на строки. Для записи в файл текстовой информации используются функции ***fputc*** и ***fputs,*** а для чтения –***fgetc*** и ***fgets.*** Строка в текстовом файле заканчивается специальным символом – ***‘\n’***.

Пример2. Переписать начальные строки (не более 10-ти) из текстового файла ***“aaa”*** в конец файла ***“bbb”***. Длина строки не превышает 80 символов (включая ***‘\n’***).

```
main()
{ FILE *f1,*f2;
 char s[81];
 int i=0;
 if((f1=fopen(“aaa”,”r”))==NULL) exit(1);
```

```
  if((f2=fopen("bbb","a"))==NULL) exit(1);
  while(fgets(s,81,f1)!=NULL && i<10) {
    i++;
    fputs(s,f2);
  }
  fclose(f1); fclose(f2);
}
```

Следует отметить, что ***char *fgets(char *s,int n,FILE *f);*** читает не более ***n-1*** литер в массив ***s,*** прекращая чтение, если встретился ***'\n'***, который включается в массив. В любом случае массив дополняется нулевым байтом. Если в решении задачи заменить второй параметр при вызове функции ***fgets*** на 80, то строки будут переписываться без искажения, но их количество может оказаться меньше десяти, если в исходном файле окажутся строки максимальной длины.

Для текстовых файлов использование функции ***int feof(FILE *f)*** имеет некоторые особенности. Если при чтении из файлового потока ***f*** достигнут конец файла, то возвращается ненулевое значение, в противном случае возвращается ноль. При этом, если не предпринималась попытка прочитать информацию за концом файла, то функция f***eof*** не будет сигнализировать о том, что достигнут конец файла.

Функции для работы со строками находятся в файле <***string.h***>. В приложении Б приведены объявления некоторых функций и их назначение.

2.2. Задание на лабораторную работу

Разработать программу обработки строки символов в соответствии с заданным вариантом. Считать, что строка оканчивается точкой, слова разделены пробелами. В программе предусмотреть ввод и вывод исходных данных и результатов, использовать функции библиотеки ***string***.

Варианты заданий приведены в таблице 2.1.

Таблица 2.1- Варианты заданий

Номер варианта	Задание
1	Во всех словах удалить первые буквы.
2	Во всех словах оставить только первые буквы.
3	Подсчитать количество слов "char" . Заменить их на "int" .
4	Подсчитать количество слов, в которых первый и последний символы различны. Однобуквенные слова не учитывать.
5	Поменять местами первое и последнее слова.
6	Удалить все цифры.
7	Найти самое длинное слово.
8	Подсчитать число слов, число символов в словах и общее число символов.
9	Подсчитать число слов, содержащих четное число символов.
10	Подсчитать количество слов, в которых первый и последний символы одинаковы. Однобуквенные слова учитывать.

3. ЛАБОРАТОРНАЯ РАБОТА №3. ИНФОРМАЦИОННЫЕ СТРУКТУРЫ ОС. ТАБЛИЦА ДЕСКРИПТОРОВ ФАЙЛОВ

Цель работы: изучение информационных структур ОС, получение навыков обработки массивов данных и отладки программ циклической структуры.

3.1. Основные теоретические положения

Для хранения информации о файлах в ОС используется таблица дескрипторов файлов.

Дескриптор файла (ДФ)– это специальная структура, связанная с каждым файлом и содержащая сведения о файле и его свойствах. Эта информация используется файловой системой для выполнения соответствующих операций с файлом. ДФ формируется тогда, когда файл создается и обновляется тогда, когда файл перемещается, редактируется или когда к нему обращаются.

Конкретное содержание ДФ зависит от ОС, но фактически во всех файловых системах в дескрипторе содержатся три основных объекта данных: идентификатор файла, сведения о том, где файл хранится, и информация, управляющая доступом.

При программировании таблица ДФ может быть представлена массивом структур. Структура – это одна или несколько переменных (возможно, различных типов), которые сгруппированы под одним именем. Эти переменные называются полями структуры. Структуры могут быть вложенными.

Описание структуры начинается с ключевого слова ***struct*** и содержит список деклараций, заключенный в фигурные скобки. За словом ***struct*** может следовать имя, называемое ***тегом структуры***. Тег дает название структуре данного вида и далее может служить кратким обозначением той части декларации, которая заключена в фигурные скобки, т. е. по сути является именем типа. Декларация структуры, не содержащей списка переменных, не резервирует памяти: она просто определяет шаблон, или образец структуры. Однако если структура имеет тег, то им далее можно пользоваться при определении структурных объектов.

Доступ к отдельному полю структуры осуществляется посредством конструкции вида: ***имя-структуры . поле-структуры*** .

Для доступа к членам структуры могут использоваться указатели. Пусть p - указатель на структуру, тогда к отдельному элементу структуры можно обратиться следующим образом: ***p-> поле –структуры*** .

Рассмотрим пример определения структуры. Пусть анкета студента представлена структурой (***struct stud***) следующего вида:

```
struct stud {
        char fio[15];          /* фамилия студента*/
        struct data { int year;
                      int mon;
                      int day;
                    } d;          /* дата рождения */
         int m[3];              /* оценки в сессию */
   };
```

Приведем функцию, которая печатает фамилии отличников и даты рождения. Параметрами функции являются массив анкет студентов ***g*** и их количество ***n***.

```
void f(struct stud g[],int n)
{ int i;
  for(i=0;i<n;i++) {
```

```
 if(g[i].m[0]==5 && g[i].m[1]==5 && g[i].m[2]==5)
        printf("%s %d.%d.%d\n",
                g[i].fio,g[i].d.day,g[i].d.mon,g[i].d.year)
 }
}
```

Одна из функций файловой системы – отображение информации о файлах в упорядоченном виде.

Сортировка – упорядочение по возрастанию (или убыванию) заданного множества элементов.

Основное условие сортировки – экономное использование памяти, т.е. перестановки элементов должны выполняться на том же месте.

Рассмотрим простые методы сортировки на примере сортировки массива ***A***, состоящего из ***n*** элементов .

Сортировка прямыми включениями:

пусть элементы $a_1, a_2, \ldots, a_{i-1}$ уже упорядочены. Очередной элемент a_i вставляется на нужное место в отсортированную часть массива. Приведем фрагмент программного кода, реализующий рассмотренный метод.

```
for (int i = 1; i< n; i++)
      {
            x=A[ i ]; j=i;
            while (x< A[ j-1 ]) &&(j>=1)
            { A[ j ]= A[ j-1];
              j=j-1; }
            A[j]= x; }
```

Сортировка прямым выбором:

выбирается наименьший элемент массива и меняется местами с первым элементом, затем просматриваются элементы, начиная со второго, и наименьший из них меняется местами (если надо) со вторым элементом, и так далее ***n-1*** раз. На последнем, ***n***-ом проходе цикла при необходимости меняются местами предпоследний и последний элементы массива. Фрагмент машинного кода приведен ниже.

```
/* просмотр массива n - 1 раз */
for (int i = 0; i< n-1; i++)
   {
      nmin= i;  min=a[i];
/* поиск наименьшего элемента */
       for (int i = i + 1; i< n; i++)
                if (A[j] < min )
                  { nmin = j; min=A[nmin]; }
     a[nmin]=A[i];
     A[i] = min;
   }
```

Сортировка прямым обменом (пузырьковая сортировка):

сравниваются пары соседних элементов, начиная с последней, и, при необходимости, элементы меняются местами. Таким образом, наименьший элемент продвигается в первую позицию. Фрагмент кода представлен ниже.

```
for (int i = 1; i< n; i++)
   for (int j = n; j>= i; j--)
```

```
if (A[ j-1]> A[ j ] )
    {    x= A[ j-1];
         A[ j-1]= A[ j] ;
         A[ j] = x;    }
```

3.2. Задание на лабораторную работу

Разработать функцию сортировки таблицы дескрипторов файлов по заданному ключу. Таблицу дескрипторов представить массивом структур. Дескриптор должен содержать имя файла (не более 8 символов), тип файла (не более 3 символов), дату создания (в формате чч.мм.гг), количество обращений (целое число), размер файла (целое число), время последней модификации(в формате час.мин). В главной программе предусмотреть ввод и вывод исходных данных и результатов, а так же обращение к функции сортировки, глобальные переменные не использовать. Варианты задания приведены в таблице 3.1.

Первый символ кода варианта задания задает ключ, второй – метод сортировки.

Таблица 3.1- Варианты задания

№ варианта	**1**	**2**	**3**	**4**	**5**	**6**
код	**11**	**12**	**13**	**21**	**22**	**23**
№ варианта	**7**	**8**	**9**	**10**	**11**	**12**
код	**31**	**32**	**33**	**41**	**42**	**43**
№ варианта	**13**	**14**	**15**	**16**	**17**	**18**
код	**51**	**52**	**53**	**61**	**62**	**63**

Тип ключа:

1. имя файла;
2. тип файла;
3. дата создания;
4. размер;
5. время последней модификации;
6. количество обращений.

Метод сортировки:

1. сортировка включениями;
2. сортировка выбором;
3. сортировка обменом.

4. ЛАБОРАТОРНАЯ РАБОТА №4. УПРАВЛЕНИЕ ДИНАМИЧЕСКИМ ВЫДЕЛЕНИЕМ ПАМЯТИ

Цель работы: изучение алгоритмов выделения памяти и программирование основных операций, выполняемых с линейными списковыми структурами.

4.1. Основные теоретические положения

Основным ресурсом системы, распределением которого занимается ОС, является оперативная память [1-4]. Память выделяется самим задачам и переменным. При ***динамическом выделении памяти*** запросы формируются во время исполнения задачи. Кроме того, при динамическом распределении требуется предоставить возможность отказываться от запрошенных блоков так, чтобы освобожденная память могла использоваться для удовлетворения последующих запросов. Таким образом, динамический распределитель должен хранить список возможно несвязных областей свободной памяти, называемый ***пулом*** или ***кучей***.

Обычно свободные блоки памяти объединяются ***в связанный список***. Поиск в списке может вестись тремя способами [1,3]:

до нахождения ***первого подходящего (first fit)*** блока,

до блока, размер которого ближе всего к заданному, т.е. ***до наиболее подходящего (best fit), ,***

до нахождения самого большого блока, ***наименее подходящего (worst fit).***

Использование стратегии ***worst fit*** имеет смысл в сочетании с сортировкой списка по убыванию размера. Это позволяет ускорить выделение памяти: всегда берется первый блок, а если он недостаточно велик, то это означает, что свободной памяти нет.

Список относится к динамическим структурам данных, которым память выделяется по мере необходимости на этапе выполнения программы. Каждый элемент списка включает, по меньшей мере, два поля: информационное, содержащее значение элемента данных, и адресное, содержащее адрес следующего элемента (информационных полей может быть несколько). Такой список называется однонаправленным (односвязным). Указатель последней записи равен коду признака конца списка, а указатель на первую запись задается дополнительным параметром, называемым переменной связи. Если в каждый элемент списка добавить еще один указатель – на предыдущий элемент, то получится двунаправленный список.

В языке С для представления элемента динамических данных используют структуру ***struct.***

Рассмотрим примеры определения элементов однонаправленного и двунаправленного списков.

Пример 1.

```
struct Node1       /* определение элемента однонаправленного списка */
{ double data;   /* data –информационное поле со значением вещественного типа*/
struct Node1 *next;};   /* next–адресное поле, содержащее указатель
                                на     следующий элемент списка */
```

Пример 2.

```
struct Node2          /* определение элемента двунаправленного списка */
{ int num;          /* num–информационное поле со значением целого типа*/
struct Node2 *next;       /* next–адресное поле, содержащее указатель
                                на       следующий элемент списка */
struct Node2 *prev; };     /* prev –адресное поле, содержащее указатель
                                на    предыдущий элемент списка */
```

Для динамического выделения и освобождения памяти в С имеются стандартные библиотечные функции:

```
#include <stdlib.h>
void *malloc (size_t size);
void *calloc (size_t nelem, size_t elsize) ;
void free (void *ptr) ;
```

Функция ***malloc()*** выделяет указанное аргументом ***size*** число байтов.
Функция ***calloc()*** выделяет память для указанного аргументом ***nelem*** числа объектов, размер которых ***elsize***. Выделенная память инициализируется нулями.
Если память не выделена, обе функции возвращают ***NULL***.
Функция ***free()*** освобождает ранее выделенную память, указатель на которую передается через аргумент ***ptr***.

В языке ***Pascal*** для представления списков используют запись ***record.***
Примеры описания списков представлены ниже.

Пример 3.

```
type                    {определение элемента однонаправленного списка}
   NLINK=^LIST;
  LIST=record
     INFO:integer; { INFO –информационное поле со значением вещественного типа }
LINK:NLINK { LINK – указатель на следующий элемент списка}
     end;
```

Пример 4.

```
type
   LINK=^LIST; { определение элемента двунаправленного списка }
  LIST=record
     INFO:integer; { INFO –информационное поле со значением вещественного типа }
    NEXT: LINK {LINK – указатель на следующий элемент списка }
    PREV: LINK { LINK – указатель на предыдущий элемент списка }
     end;
```

Для динамического выделения и освобождения памяти в языке ***Pascal*** используются процедуры ***new и dispose***:

```
new ( var P:ptr );  {выделяет память указанного типа и возвращает адрес в P}
dispose( var P:ptr );  {освобождает память, на которую указывает указанного P}
```

4.2. Задание на лабораторную работу

Разработать подсистему динамического выделения памяти, язык программирования С. Для отладки разработать программу, в которой предусмотреть формирование списков свободной и выделенной памяти, имитацию запроса и выполнения заданной операции. Элемент списка должен содержать адрес первого байта блока памяти и размер блока. В случае, когда освобождающийся блок примыкает к соседнему блоку, он должен объединяться с ним в один блок. В программе обеспечить вывод исходных данных и результатов.

Главная программа должна формировать меню, содержащее следующие пункты:

ПОКАЗАТЬ СОСТОЯНИЕ ПАМЯТИ,
ВЫДЕЛИТЬ ПАМЯТЬ,
ОСВОБОДИТЬ ПАМЯТЬ,
ВЫХОД.

Коды вариантов приведены в таблице 4.1. Первая компонента кода варианта задания задает стратегию выделения памяти, вторая – метод сортировки, третья –тип списка.

Таблица 4.1– Варианты задания.

№ варианта	**1**	**2**	**3**	**4**	**5**	**6**
Код	**111**	**112**	**121**	**122**	**131**	**132**
№ варианта	**7**	**8**	**9**	**10**	**11**	**12**
Код	**211**	**212**	**221**	**222**	**231**	**232**
№ варианта	**13**	**14**	**15**	**16**	**17**	**18**
Код	**311**	**312**	**321**	**322**	**331**	**332**

Стратегия выделения памяти:

1. первый подходящий;
2. наиболее подходящий;
3. наименее подходящий.

Метод сортировки:

1. сортировка выбором;
2. сортировка обменом;
3. сортировка включениями.

Тип списка:

1. однонаправленный;
2. двунаправленный.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Таненбаум Э. Современные операционные системы. / Э. Таненбаум. –СПб.: Питер, 2011. –1120 с.
2. Гордеев А. В. Операционные системы. / А. В. Гордеев – СПб.: Питер, 2007. –416 с.
3. Иртегов Д. В. Введение в операционные системы. / Д. В. Иртегов – СПб.: БХВ-Петербург, 2002. – 624 с.
4. Столлингс В. Операционные системы. / В. Столлингс. – М.: Издательский дом «Вильямс», 2002. –848 с.
5. Олифер В.Г. Сетевые операционные системы /Олифер В.Г., Олифер Н.А. – СПб.: Питер, 2001. – 544 с.