МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Алгоритмы и структуры данных»

Тема: Программирование рекурсивных алгоритмов

Студентка гр. 9303 _____ Булыно Д. А.

Преподаватель _____ Филатов А. Ю.

Санкт-Петербург

2020

Цель работы.

Формирование практических навыков программирования рекурсивных алгоритмов на языке программирования C++ путём решения поставленной задачи.

Основные теоретические положения.

Рекурсия — это такой способ организации вспомогательного алгоритма (подпрограммы), при котором эта подпрограмма (процедура или функция) в ходе выполнения ее операторов обращается сама к себе. Вообще, рекурсивным называется любой объект, который частично определяется через себя.

В рекурсивном определении должно присутствовать ограничение, граничное условие, при выходе на которое дальнейшая инициация рекурсивных обращений прекращается.

Обращение к рекурсивной подпрограмме ничем не отличается от вызова любой другой подпрограммы. При этом при каждом новом рекурсивном обращении в памяти создаётся новая копия подпрограммы со всеми локальными переменными. Такие копии будут порождаться до выхода на граничное условие. Очевидно, в случае отсутствия граничного условия, неограниченный рост числа таких копий приведёт к аварийному завершению программы за счёт переполнения стека.

Порождение все новых копий рекурсивной подпрограммы до выхода на граничное условие называется рекурсивным спуском. Максимальное количество копий рекурсивной подпрограммы, которое одновременно может находиться в памяти компьютера, называется глубиной рекурсии. Завершение работы рекурсивных подпрограмм, вплоть до самой первой, инициировавшей рекурсивные вызовы, называется рекурсивным подъёмом.

Использование рекурсии является «красивым» приёмом программирования. В то же время в большинстве практических задач этот приём неэффективен с точки зрения расходования таких ресурсов ЭВМ, как память и

время исполнения программы. Использование рекурсии увеличивает время исполнения программы и зачастую требует значительного объёма памяти для хранения копий подпрограммы на рекурсивном спуске. Поэтому на практике разумно заменять рекурсивные алгоритмы на итеративные.

Задание.

Вариант 4

Напечатать все перестановки заданных *n* различных натуральных чисел (или символов).

Выполнение работы.

Для выполнения лабораторной работы были реализованы следующие функции:

1. `void sort_int(int *a, int *b)`
2. `void print_int(int a[], int n)`
3. `int *Ost(int a[], int n, int &j)`
4. `int *Res(int ost_1[], int ost_2[], int k, int j)`
5. `int *Alm_perm(int a[], int res[], int n, int j)`
6. `void swap_int(int alm_perm[], int n, int j)`
7. `int *sort_Ost(int alm_perm[], int n, int j)`
8. `int *Perm(int alm_perm[], int sort_ost[], int n, int j)`
9. `void delete_all(int ost[], int ost_1[], int ost_2[], int res[], int alm_perm[], int sort_ost[], int perm[])`
10. `int perms_int(int a[], int n)`
11. `void sort_symb(char a[], int n)`
12. `void print_symb(char a[], int n)`
13. `char *Ost_symb(char a[], int n, int &j)`
14. `char *Res_symb(char ost_1[], char ost_2[], int k, int j)`
15. `char *Alm_perm_symb(char a[], char res[], int n, int j)`
16. `void swap_symb(char alm_perm[], int n, int j)`
17. `char *sort_Ost_symb(char alm_perm[], int n, int j)`
18. `int *Perm_symb(char alm_perm[], char sort_ost[], int n, int j)`

19. `void delete_all_symb(char ost[], char ost_1[], char ost_2[], char res[], char alm_perm[], char sort_ost[], char perm[])`
20. `int perms_symb(char a[], int n)`
21. `void Error(short n)`

Функция `void sort_int(int *a, int *b)` принимает на вход 2 указателя на объект типа `int`. Функция производит сортировку целочисленного массива в порядке возрастания. Функция ничего не возвращает.

Функция `void print_int(int a[], int n)` принимает на вход целочисленный массив и количество элементов данного массива. Функция выводит данный целочисленный массив. Функция ничего не возвращает.

Функция `int *Ost(int a[], int n, int &j)` принимает на вход целочисленный массив, количество элементов исходного массива и ссылку на объект типа `int` для количества элементов нового массива `ost`. Функция находит упорядоченный по убыванию «остаток» перестановки наибольшей длины. Функция возвращает этот «остаток» в порядке возрастания.

Функция `int *Res(int ost_1[], int ost_2[], int k, int j)` принимает на вход 2 целочисленных массива (части массива `ost`), количество элементов массива `ost_1` и количество элементов массива `ost`. Функция формирует новый массив `res` из массивов `ost_1` и `ost_2`. Функция возвращает новый массив.

Функция `int *Alm_perm(int a[], int res[], int n, int j)` принимает на вход 2 целочисленных массива (исходный массив `a` и массив `res`), количество элементов массива `a` и количество элементов массива `ost`. Функция формирует новый массив `alm_perm` из массивов `a` и `res`. Функция возвращает новый массив.

Функция `void swap_int(int alm_perm[], int n, int j)` принимает на вход целочисленный массив, количество элементов исходного массива и количество элементов массива `ost`. Функция меняет элемент, предшествующий «остатку», с наименьшим элементом «остатка», большим данного элемента. Функция ничего не возвращает.

Функция `int *sort_Ost(int alm_perm[], int n, int j)` принимает на вход целочисленный массив, количество элементов исходного массива и количество

элементов массива ost. Функция сортирует «остаток» в порядке возрастания. Функция возвращает отсортированный массив.

Функция int *Perm(int alm_perm[], int sort_ost[], int n, int j) принимает на вход 2 целочисленных массива (массив alm_perm и массив sort_ost), количество элементов исходного массива и количество элементов массива ost. Функция формирует новый массив perm из массивов alm_perm и sort_ost, в результате чего получается перестановка. Функция возвращает новый массив.

Функция void delete_all(int ost[], int ost_1[], int ost_2[], int res[], int alm_perm[], int sort_ost[], int perm[]) удаляет все образованные динамические массивы. Функция ничего не возвращает.

Функция int perms_int(int a[], int n) принимает на вход исходный целочисленный массив и его количество элементов. Функция рекурсивно ищет все перестановки данного массива. С помощью функции Ost() функция находит упорядоченный по убыванию «остаток» перестановки наибольшей длины и записывает в новый массив ost. Если количество элементов «остатка» равно количеству элементов исходного массива, функция возвращает 0. Далее функция ищет наименьший элемент в «остатке», который больше элемента, предшествующего «остатку». Для этого в массив ost_1 записываются все элементы «остатка», которые больше элемента, предшествующего «остатку», а в массив ost_2 - элементы «остатка», которые меньше элемента, предшествующего «остатку». С помощью функции Res() функция формирует новый массив из массивов ost_1 и ost_2 и записывает в новый массив res. С помощью функции Alm_perm() функция формирует новый массив из массивов a и res и записывает в новый массив alm_perm. С помощью функции swap_int() функция меняет элемент, предшествующий «остатку», с наименьшим элементом «остатка», большим данного элемента. С помощью функции sort_Ost() функция сортирует «остаток» в порядке возрастания и записывает в новый массив sort_ost. С помощью функции Perm() функция формирует новый массив perm из массивов alm_perm и sort_ost, в результате чего получается новая перестановка.

С помощью функции `print_int()` функция выводит перестановку. Функция вызывается рекурсивно. После этого с помощью функции `delete_all()` функция удаляет все образованные динамические массивы. Функция возвращает 1.

Функции 11-20, в названии которых присутствует `symb`, служат для обработки символьного массива, они идентичны функциям для обработки целочисленного массива.

Функция `void Error(short n)` принимает на вход число. В зависимости от этого числа функция выводит ошибку определённого типа.

В функции `int main()` производится ввод-вывод массивов, обработка ошибок и вызов рекурсивных функций.

Выводы.

В ходе выполнения лабораторной работы получены практические навыки программирования рекурсивных алгоритмов на языке программирования C++. В результате выполнения лабораторной работы была написана программа, которая печатает все перестановки заданных n различных натуральных чисел (или символов).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <cctype>
#include <cstdlib>
#include <cstring>
#include <string>
using namespace std;

void sort_int(int *a, int *b){
	bool state = false;
	int *c = b - 1;
	for (int *i = a; i < c; i++){
		if (*i > *(i + 1)){
			swap(*i, *(i + 1));
			state = true;
		}
	}
	if (state) sort_int(a, b);
}

void print_int(int a[], int n){
	for(int i = 0; i < n; i++)
		cout << a[i] << " ";
	cout << endl;
}

int *Ost(int a[], int n, int &j){
	int *ost = new int[j];
	for(int i = n-1; i >= 0; i--){
		if(a[i] > a[i-1]){
			ost[n-1-i]=a[i];
			j++;
			break;
		}else{
			ost[n-1-i]=a[i];
			j++;
		}
	}
	return ost;
}

int *Res(int ost_1[], int ost_2[], int k, int j){
```

```
	int p = 0;
	int t = 0;
	int *res = new int[j];
	for (int i = 0; i < j; i++){
		if (i < k){
			res[i] = ost_1[p];
			p++;
		}else{
			res[i] = ost_2[t];
			t++;
		}
	}
	return res;
}

int *Alm_perm(int a[], int res[], int n, int j){
	int r = 0;
	int *alm_perm = new int[n];
	for (int i = 0; i < n; i++){
		if(i < n-j){
			alm_perm[i] = a[i];
		}else{
			alm_perm[i] = res[r];
			r++;
		}
	}
	return alm_perm;
}

void swap_int(int alm_perm[], int n, int j){
	for (int i = 0; i < n; i++){
		if (i == n-j-1){
			int temp = alm_perm[i];
			alm_perm[i] = alm_perm[i+1];
			alm_perm[i+1] = temp;
		}
	}

}

int *sort_Ost(int alm_perm[], int n, int j){
	int k = 0;
	int *sort_ost = new int[j];
	for(int i = 0; i < n; i++){
```

```
            if(i >= n-j){
                  sort_ost[k] = alm_perm[i];
                  k++;
            }
      }
      sort_int(sort_ost, sort_ost+j);
      return sort_ost;
}

int *Perm(int alm_perm[], int sort_ost[], int n, int j){
      int l = 0;
      int *perm = new int[n];
      for (int i = 0; i < n; i++){
            if (i < n-j){
                  perm[i] = alm_perm[i];
            }else{
                  perm[i] = sort_ost[l];
                  l++;
            }
      }
      return perm;
}

void delete_all(int ost[], int ost_1[], int ost_2[], int res[], int
alm_perm[], int sort_ost[], int perm[]){
      delete [] ost;
      delete [] ost_1;
      delete [] ost_2;
      delete [] res;
      delete [] alm_perm;
      delete [] sort_ost;
      delete [] perm;
}

int perms_int(int a[], int n){
      int j = 0;
      int min = 0;

      int *ost = new int[n-1];
      ost = Ost(a, n, j);

      if (j == n) return 0;

      int k = 0;
```

```
	int m = 0;

	int *ost_1 = new int[j];
	int *ost_2 = new int[j];
	for (int i = 0; i < j; i++){
		if(ost[i] > a[n-j-1]){
			ost_1[k] = ost[i];
			k++;
		}else{
			ost_2[m] = ost[i];
			m++;
		}
	}

	int *res = new int[j];
	res = Res(ost_1, ost_2, k, j);

	int *alm_perm = new int[n];
	alm_perm = Alm_perm(a, res, n, j);

	swap_int(alm_perm, n, j);

	int *sort_ost = new int[j];
	sort_ost = sort_Ost(alm_perm, n, j);

	int *perm = new int[n];
	perm = Perm(alm_perm, sort_ost, n, j);

	print_int(perm, n);

	perms_int(perm, n);

	delete_all(ost, ost_1 ,ost_2, res, alm_perm, sort_ost, perm);

	return 1;
}

void sort_symb(char a[], int n){
	char temp = a[0];
	for (int i = 0; i < n; i++){
		for(int j = 0; j < n; j++){
			if (tolower(a[i]) < tolower(a[j])){
				temp = a[i];
				a[i] = a[j];
```

```
                    a[j] = temp;
                }
            }
        }

}

void print_symb(char a[], int n){
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
    cout << endl;
}

char *Ost_symb(char a[], int n, int &j){
    char *ost = new char[j];
    for(int i = n-1; i >= 0; i--){
        if(a[i] > a[i-1]){
            ost[n-1-i]=a[i];
            j++;
            break;
        }else{
            ost[n-1-i]=a[i];
            j++;
        }
    }
    return ost;
}

char *Res_symb(char ost_1[], char ost_2[], int k, int j){
    int p = 0;
    int t = 0;
    char *res = new char[j];
    for (int i = 0; i < j; i++){
        if (i < k){
            res[i] = ost_1[p];
            p++;
        }else{
            res[i] = ost_2[t];
            t++;
        }
    }
    return res;
}
```

```
char *Alm_perm_symb(char a[], char res[], int n, int j){
	int r = 0;
	char *alm_perm = new char[n];
	for (int i = 0; i < n; i++){
		if(i < n-j){
			alm_perm[i] = a[i];
		}else{
			alm_perm[i] = res[r];
			r++;
		}
	}
	return alm_perm;
}

void swap_symb(char alm_perm[], int n, int j){
	for (int i = 0; i < n; i++){
		if (i == n-j-1){
			char temp = alm_perm[i];
			alm_perm[i] = alm_perm[i+1];
			alm_perm[i+1] = temp;
		}
	}

}

char *sort_Ost_symb(char alm_perm[], int n, int j){
	char *sort_ost = new char[j];
	int k = 0;
	for(int i = 0; i < n; i++){
		if(i >= n-j){
			sort_ost[k] = alm_perm[i];
			k++;
		}
	}
	sort_symb(sort_ost, j);
	return sort_ost;
}

char *Perm_symb(char alm_perm[], char sort_ost[], int n, int j){
	int l = 0;
	char *perm = new char[n];
	for (int i = 0; i < n; i++){
		if (i < n-j){
			perm[i] = alm_perm[i];
```

```
            }else{
                  perm[i] = sort_ost[l];
                  l++;
            }
      }
      return perm;
}

void delete_all_symb(char ost[], char ost_1[], char ost_2[], char res[],
char alm_perm[], char sort_ost[], char perm[]){
      delete [] ost;
      delete [] ost_1;
      delete [] ost_2;
      delete [] res;
      delete [] alm_perm;
      delete [] sort_ost;
      delete [] perm;
}

int perms_symb(char a[], int n){
      int j = 0;
      int min = 0;
      char *ost = new char[n-1];
      ost = Ost_symb(a, n, j);

      if (j == n) return 0;

      int k = 0;
      int m = 0;

      char *ost_1 = new char[j];
      char *ost_2 = new char[j];
      for (int i = 0; i < j; i++){
            if(ost[i] > a[n-j-1]){
                  ost_1[k] = ost[i];
                  k++;
            }else{
                  ost_2[m] = ost[i];
                  m++;
            }
      }

      char *res = new char[j];
      res = Res_symb(ost_1, ost_2, k, j);
```

```
	char *alm_perm = new char[n];
	alm_perm = Alm_perm_symb(a, res, n, j);

	swap_symb(alm_perm, n, j);

	char *sort_ost = new char[j];
	sort_ost = sort_Ost_symb(alm_perm, n, j);

	char *perm = new char[n];
	perm = Perm_symb(alm_perm, sort_ost, n, j);

	print_symb(perm, n);

	perms_symb(perm, n);

	delete_all_symb(ost, ost_1 ,ost_2, res, alm_perm, sort_ost, perm);

	return 1;
}

void Error(short n){
	cout << "Error #" << n << ". ";
	switch (n){
		case 0:
			cout << "Размер массива не задан." << endl;
			break;
		case 1:
			cout << "Тип массива не определён." << endl;
			break;
		case 2:
			cout << "Массив не состоит из натуральных чисел." <<
endl;
			break;
		case 3:
			cout << "Массив не состоит из символов." << endl;
			break;
		case 4:
			cout << "Массив состоит из повторяющихся значений." <<
endl;
			break;
		default:
			cout << "! .";
			break;
```

```
	}
}

int main(){
	int n;
	cout << "Введите размер массива: n = ";
	cin >> n;

	if (!n){
		Error(0);
		return 0;
	}

	string type;
	cout << "Введите тип массива ('0' для числового массива, 'a' - для
символьного): ";
	cin >> type;

	if (type == "0"){
		int a[n];
		cout << "Введите массив из чисел через пробел: a[n] = ";
		for (int i = 0; i < n; i++){
			cin >> a[i];
			if (a[i] > 0) continue;
			else{
				Error(2);
				return 0;
			}
		}

		sort_int(a, a+n);
		for (int i = 0; i < n; i++){
			if (a[i] == a[i+1]){
				Error(4);
				return 0;
			}
		}

		cout << "Перестановки данного массива: " << endl;
		print_int(a, n);
		perms_int(a, n);
	}

	else if(type == "a" || type == "A"){
```

```
            char a[n];
            cout << "Введите массив из символов: a[n] = ";
            for (int i = 0; i < n; i++){
                  cin >> a[i];
                  if (isalpha(a[i])) continue;
                  else{
                        Error(3);
                        return 0;
                  }
            }

            sort_symb(a, n);

            for (int i = 0; i < n; i++){
                  if (a[i] == a[i+1]){
                        Error(4);
                        return 0;
                  }
            }

            cout << "Перестановки данного массива: " << endl;
            print_symb(a, n);
            perms_symb(a, n);
      }

      else{
            Error(1);
            return 0;
      }
}
```

ПРИЛОЖЕНИЕ Б

РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

Таблица 1. Результаты работы программы

№	Входные данные	Результат
1.	n = 3 type = 0 a[n] = 3 8 6	3 6 8 3 8 6 6 3 8 6 8 3 8 3 6 8 6 3
2.	n = 3 type = a a[n] = d m e s	d e m s d e s m d m e s d m s e d s e m d s m e e d m s e d s m e m d s e m s d e s d m e s m d m d e s m d s e m e d s m e s d m s d e m s e d s d e m s d m e s e d m s e m d s m d e s m e d
3.	n = 2 type = A a[n] = a s	a s s a
4.	n = a	Error #0. Размер массива не задан.

5.	n = 5 type = r	Error #1. Тип массива не определён.
6.	n = 5 type = 0 a[n] = -4 k 0 3 5	Error #2. Массив не состоит из натуральных чисел.
7.	n = 5 type = a a[n] = -4 k 0 3 5	Error #3. Массив не состоит из символов.
8.	n = 5 type = a a[n] = a d e s d	Error #4. Массив состоит из повторяющихся значений.
9.	n = 5 type = 0 a[n] = 1 4 3 2 3	Error #4. Массив состоит из повторяющихся значений.