МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. Шухова»
(БГТУ им. В. Г. Шухова)

Кафедра программного обеспечения вычислительной
техники и автоматизированных систем

Лабораторная работа №19.9
по дисциплине: «Использование функций
при решении задач на одномерные массивы»

Выполнил/а: ст. группы ВТ-231
Кисиль Николай Владимирович

Проверили:
Черников Сергей Викторович
Новожен Никита Викторович

Белгород, 2023 г.

Цель работы: получение навыков решении задач на одномерные массивы.

Содержание работы

Задача 1: Если возможно, то упорядочить данный массив размера n по убыванию, иначе массив оставить без изменения.

Пример тестовых данных:

№	Входные данных	Выходные данные
1	1 2 4	4 2 1
2	4 2 4	4 2 4
3	1 3 1 4	1 3 1 4
4	4 2 3 1	4 3 2 1

Спецификация функции selectionSortDecreasing:

1. Заголовок `void selectionSortDecreasing(int *a, const size_t n)`
   Назначение: если возможно, то упорядочить данный массив размера n по убыванию, иначе массив оставить без изменения.

Код:

```
void selectionSortDecreasing(int *a, const size_t n) {
    int can_sorted = 1;
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < i; j++) {
            if(a[i] == a[j]) {
                can_sorted = 0;
            }
        }
    }
    if(can_sorted) {
        selectionSort(a, n);
    }
    outputArray(a, n);
}
```

Задача 2: Дана целочисленная последовательность. Упорядочить по неубыванию часть последовательности, заключенную между первым вхождением максимального значения и последним вхождением минимального.

Пример тестовых данных:

№	Входные данных	Выходные данные
1	10 3 2 1 0	10 1 2 3 0
2	0 3 2 1 10	0 1 2 3 10
3	10 5 4 4 7 8 10 10	10 4 5 4 7 8 10 10

Спецификация функции maxIndex:

1. Заголовок `int maxIndex(const int *a, const size_t n)`
2. Назначение: поиск позиции максимального значения элемента

Спецификация функции maxIndex:

1. Заголовок `int minIndex(const int *a, const size_t n)`
2. Назначение: поиск позиции минимального значения элемента

Спецификация функции selectionSortOfPart:

1. Заголовок `void selectionSortOfPart(int *a, size_t beginIndex, size_t endIndex)`
2. Назначение: сортировка части массива

Спецификация функции sortBetweenFirstMaxLastMin:

1. Заголовок `void sortBetweenFirstMaxLastMin(int *a, size_t n)`
2. Назначение: упорядочить по неубыванию часть последовательности, заключенную между первым вхождением максимального значения и последним вхождением минимального

Код:

```
int maxIndex(const int *a, const size_t n) {
    int max = 0;
    for (int i = 0; i < n; i++) {
        if(a[i] > a[max])
            max = i;
    }
    return max;
}

int minIndex(const int *a, const size_t n) {
    int min = 0;
    for (int i = 0; i < n; i++) {
        if(a[i] <= a[min])
            min = i;
    }
    return min;
}

void selectionSortOfPart(int *a, size_t beginIndex, size_t endIndex) {
    selectionSort(a + beginIndex, endIndex - beginIndex);
}

void sortBetweenFirstMaxLastMin(int *a, size_t n) {
    int min = minIndex(a, n) + 1;
    int max = maxIndex(a, n);
    int need_swap = min - 1 > max;

    sort2(&min, &max);
    selectionSortOfPart(a, max, min);

    if(need_swap) {
        swap(&a[min - 1], &a[max]);
    }
}
```

Задача 3: Если данная последовательность не упорядочена ни по неубыванию, ни по невозрастанию, найти среднее геометрическое положительных членов

Пример тестовых данных:

№	Входные данных	Выходные данные
1	4 1 2	2
2	9 1 3 3 0	3
3	2 -1 -1 0	2
4	-1 -2 -1	0
5	2 4 3	2.884499
6	-1 -1 -1	"Последовательность упорядочена"
7	1 2 4	"Последовательность упорядочена"
8	4 2 2	"Последовательность упорядочена"

Спецификация функции isSortedUnDecreasing:

1. Заголовок `int isSortedUnDecreasing(int *a, size_t n)`
2. Назначение: возвращает «истина» если последовательность упорядочена по неубыванию, в противном случае – «ложь».

Спецификация функции isSortedUnIncreasing:

1. Заголовок `int isSortedUnIncreasing(int *a, size_t n)`
2. Назначение: возвращает «истина» если последовательность упорядочена по невозрастанию, в противном случае – «ложь».

Спецификация функции geometricMean:

1. Заголовок `double geometricMean(const int *a, size_t n)`
2. Назначение: возвращает среднее геометрическое от чисел

Спецификация функции checkSort:

1. Заголовок `void checkSort(int * a, size_t n)`
2. Назначение: Если данная последовательность не упорядочена ни по неубыванию, ни по невозрастанию, вовзращает среднее геометрическое положительных членов

Код:

```
int isSortedUnDecreasing(int *a, size_t n) {
    for(size_t i = 1; i < n; i++) {
        if(a[i] < a[i - 1]) {
            return 0;
        }
    }
    return 1;
}


int isSortedUnIncreasing(int *a, size_t n) {
    for(size_t i = 1; i < n; i++) {
        if(a[i] > a[i - 1]) {
            return 0;
        }
    }
    return 1;
}

double geometricMean(const int *a, size_t n) {
    int comp = 1;
    float count = 0;
    for(size_t i = 0; i < n; i++) {
        if(a[i] > 0) {
            count++;
            comp *= a[i];
        }
    }
    if(count == 0)
        comp = 0;

    return pow(comp, (float ) 1/count);
}

void checkSort(int * a, size_t n) {
    if(isSortedUnDecreasing(a, n) || isSortedUnIncreasing(a, n)) {
        printf("Последовательность упорядочена");
    } else {
        printf("%f", geometricMean(a, n));
    }
}
```

Задача 4 Если число x встречается в данной целочисленной последовательности, то упорядочить по неубыванию часть последовательности после первого вхождения x.

Пример тестовых данных:

№	Входные данных	Выходные данные
1	16 8 4 2 1 4	16 8 4 1 2
2	16 8 4 2 1 16	16 1 2 4 8
3	16 8 4 2 1 1	16 8 4 2 1
4	16 8 4 2 1 3	16 8 4 2 1

Спецификация функции sortAfter:

3. Заголовок `void sortAfter(const int *a, const size_t n, const int x)`
4. Назначение: упорядочивает по неубыванию часть последовательности после первого вхождения x, если x встречается в данной последовательности.

Код:

```
void sortAfter(const int *a, const size_t n, const int x) {
    int pos = getIndex(a, n, x);
    if(pos >= 0) {
        if(pos != n - 1)
            pos++;
        selectionSortOfPart(a, pos, n);
    }
    outputArray(a, n);
}
```

Задача 5: Даны две последовательности. Получить упорядоченную по невозрастанию последовательность, состоящую из тех членов первой последовательности, которых нет во второй.

Пример тестовых данных:

№	Входные данных	Выходные данные
1	1 2 4 4	2 1
2	1 2 2 4 1 4	2 2
3	1 2 2 4 3	4 2 2 1
4	1 3 3 4 6 8 8 9 10 12 12 13 15 15 0 3 5 5 8 9 9 11 14 14	1 4 6 10 12 12 13 15 15

Спецификация функции sortException:

1. Заголовок void sortException(int *a, size_t an, int *b, size_t bn)
2. Назначение: возвращает упорядоченную по невозрастанию последовательность, состоящую из тех членов первой последовательности, которых нет во второй.

Код:

```
void sortException(int *a, size_t an, int *b, size_t bn) {
    selectionSort(a, an);
    selectionSort(b, bn);
    if(an > bn) {
        int c[an];
        int j = 0;
        for(size_t i = 0; i < an; i++) {
            if(getIndex(b, bn, a[i]) < 0) {
                c[j] = a[i];
                j++;
            }
        }
        outputArray(c, j);
    } else {
        int c[bn];
        int j = 0;
        for(size_t i = 0; i < an; i++) {
            if(getIndex(a, an, b[i]) < 0) {
                c[j] = b[i];
                j++;
            }
        }
        outputArray(c, j);
    }
}
```

Задача 6: Дана целочисленная последовательность, содержащая как положительные, так и отрицательные числа. Упорядочить последовательность следующим образом: сначала идут отрицательные числа, упорядоченные по невозрастанию, потом положительные, упорядоченные по неубыванию.

Пример тестовых данных:

№	Входные данных	Выходные данные
1	3 2 1 1 -4 -5 -6	-4 -5 -6 1 1 2 3
2	-3 -2 -1 0 1 2 3 4	-1 -2 -3 0 1 2 3 4
3	1 2 3 4 5	1 2 3 4 5

Спецификация функции orderSequence:

1. Заголовок void orderSequence(int * a, int n)
2. Назначение: упорядочивает так, что сначала идут отрицательные числа, упорядоченные по невозрастанию, потом положительные, упорядоченные по неубыванию.

Код:

```
void orderSequence(int * a, int n) {

    int neg_count = 0;
    int pos_count = 0;
    for (int i = 0; i < n; i++) {
        if (a[i] < 0) {
            neg_count++;
        } else {
            pos_count++;
        }
    }

    int neg[n];
    int pos[n];

    int neg_index = 0;
    int pos_index = 0;
    for (int i = 0; i < n; i++) {
        if (a[i] < 0) {
            neg[neg_index++] = a[i];
        } else {
            pos[pos_index++] = a[i];
        }
    }

    if(neg_count) selectionSortIncreasing(neg, neg_count);
    if(pos_count) selectionSort(pos, pos_count);

    neg_index = 0;
    pos_index = 0;

    for (int i = 0; i < n; i++) {
        if (neg_index < neg_count) {
            a[i] = neg[neg_index++];
        } else {
            a[i] = pos[pos_index++];
        }
    }
}
```

Задача 7: Дана целочисленная последовательность (по определению содержащая как положительные, так и отрицательные элементы) и целое число x. Определить, есть ли x среди членов последовательности, и если нет, то найти члены последовательности, ближайшие к x снизу и сверху.

Пример тестовых данных:

№	Входные данных	Выходные данные
1	1 3 6 2 5 6	«x - элемент последовательности»
2	1 3 6 2 5 4	3 5
3	1 3 6 2 5 0	$-\infty$, 1
4	1 3 6 2 5 7	6 ∞
5	1 1 5 5 5 3	1 5

Спецификация функции nearestBelowAndAbove:

1. Заголовок void nearestBelowAndAbove(int *a, int n, int x)
2. Назначение если x нет среди членов последовательности, то найти члены последовательности, ближайшие к x снизу и сверху.

Код:

```
void nearestBelowAndAbove(int *a, int n, int x) {
    int pos = getIndex(a, n, x);
    if(pos >= 0) {
        printf("x элемент последовательности");
    } else {
        int below[n];
        int above[n];
        int j, k = 0;
        for(size_t i = 0; i < n; i++) {
            if(a[i] - x > 0 && a[i] > x) {
                above[j] = a[i];
                j++;
            } else {
                below[k] = a[i];
                k++;
            }
        }
        int below_index = maxIndex(below, k);
        int above_index = minIndex(above, j);

        if(getIndex(a, n, below[below_index]) < 0) {
            printf("-∞ %d", above[above_index]);
        } else if(getIndex(a, n, above[above_index]) < 0) {
            printf("%d ∞", below[below_index]);
        } else {
            printf("%d %d", below[below_index], above[above_index]);
        }
    }
}
```

Задача 8: Дана целочисленная последовательность. Получить массив из уникальных элементов последовательности.

Пример тестовых данных:

№	Входные данных	Выходные данные
1	1 2 4 1 2	4
2	1 2 3 4 5	1 2 3 4 5
3	1 1 1 1 1	

Спецификация функции saveUniqueOnce:

1. Заголовок void saveUniqueOnce(int *a, int n)
2. Назначение: возвращает массив из уникальных элементов последовательности

Код:

```
void saveUniqueOnce(int *a, int n) {
    int b[n];
    int j = 0;
    for(size_t i = 0; i < n; i++) {
        int pos = getIndex(a, n, a[i]);
        if(getIndex(a+pos+1, n, a[i]) < 0) {
            b[j] = a[i];
            j++;
        }
    }
    outputArray(b, j);
}
```

Задача 9: Определить, можно ли, переставив члены данной целочисленной последовательности длины n ($n > 1$), получить геометрическую прогрессию с знаменателем q ($|q| \neq 1$). Разрешимое допущение: знаменатель прогрессии – целое число.

Пример тестовых данных:

№	Входные данных	Выходные данные
1	4 1 2	“YES”
2	-1 -4 -16 2 8	“YES”
3	1 2 5	“NO”
4	1 1	“NO”
5	0 1	“NO”
6	1 3 0	“NO”
7	1 2 -4 -8 -16	“NO”
8	1 1 1 1 -1	“NO”
9	0 0 0	“NO”
10	1 2 4 4 4 4 8	“NO”
11	1 -1 1	“NO”

Спецификация функции selectionSortAbs:

1. Заголовок void selectionSortAbs(int *a, const size_t n)
2. Назначение: упорядочивает массив по модулю значений элементов

Спецификация функции isGeometricProgression:

1. Заголовок void isGeometricProgression(int *a, size_t n)
2. Назначение: возвращает «YES» если возможно составить геометрическую прогрессию, и, если нет – «NO»

Код:

```
void selectionSortAbs(int *a, const size_t n) {
    for (int i = 0; i < n - 1; i++) {
        int min_index = i;
        for (int j = i + 1; j < n; j++)
            if (abs(a[j]) < abs(a[min_index]))
                min_index = j;
        swap(&a[i], &a[min_index]);
    }
}


void isGeometricProgression(int *a, size_t n) {
    selectionSortAbs(a, n);
    float q = (float) a[1] / (float) a[0];
    int is_progression = 1;
    for(size_t i = 0; i < n - 1; i++) {
        if((float) a[i+1] / (float) a[i] != q) {
            is_progression = 0;
        }
    }
    if(is_progression && q != 1) {
        printf("YES");
    } else {
        printf("NO");
    }
}
```

Задача 10: **Найти сумму четных цифр элементов массива из положительных чисел.

Пример тестовых данных:

№	Входные данных	Выходные данные
1	1 2 3 4 5	6
2	11 12 13 14 15	6
3	10 -12 22 -26 101	4
4	-12 -14 -16 -8 -4	0

Спецификация функции getSumOfEvenDigits:

1. Заголовок int getSumOfEvenDigits(long long x)
2. Назначение: возвращает сумму четных цифр числа

Спецификация функции sumEvenDigitsPosNumbers:

1. Заголовок void sumEvenDigitsPosNumbers (int *a, size_t n)
2. Назначение: возвращает сумму четных цифр положительных элементов массива

Код:

```
int getSumOfEvenDigits(long long x) {
    int sum = 0;
    while (x > 0) {
        if(isEven(x)) {
            sum += x % 10;
        }
        x /= 10;
    }
    return sum;
}

int sumEvenDigitsPosNumbers(int *a, size_t n) {
    int sum = 0;
    for(size_t i = 0; i < n; i++) {
        if(isPositive(a[i])) {
            sum += getSumOfEvenDigits(a[i]);
        }
    }
    return sum;
}
```

Вывод: получили навыки использования функция для решения задач на С.