Тренин С.А. Информатика. Осень 2014.

Практикум 14. Поиск минимума и максимума. Сортировка выбором. Указатели на функцию. Массивы указателей на функцию.

1. Напишите программу сортировки массива из N (1..10 000) целых чисел из диапазона -1 000 000 до +1 000 000 по возрастанию. (Используйте алгоритм сортировки выбором минимального элемента).

   В отдельной функции организуйте ввод элементов массива:

```
void InputArray (int input_data[], int length);
```

   В отдельной функции организуйте поиск индекса минимального элемента на подмассиве:

```
int FindMinimumIndex (int input_data[], int length,
                      int first_index, int last_index);
```

   В отдельной функции реализуйте обмен элементов массива.

```
void Swap (int &first_element, int &second_element);
```

   В отдельной функции организуйте сортировку.

```
void SelectionSort (int input_data[], int length);
```

   В отдельной функции организуйте вывод элементов упорядоченного массива:

```
void OutputArray (int input_data[], int length);
```

Ваша программа должна считывать входные данные из входного потока. Формат входного файла следующий: в первой строчке количество элементов массива, в следующих строчках сами элементы не отсортированного массива по одному на строчку. Программа должна выводить в стандартный выходной поток отсортированный массив в том же формате.

Пример входного файла:	Пример соответствующего выходного файла:
5 100 0 501 500 32768	5 0 100 500 501 32768

Протестируйте работоспособность алгоритма на разных массивах.

2. Везде где возможно замените сравнение на функцию:

```
int Compare (void *first_pointer, void *second_pointer ) {
     int *first_element_int_pointer = (int *)first_pointer;
     int *second_element_int_pointer = (int *)second_pointer;
     return *first_element_int_pointer –
                                   *second_element_int_pointer;
}
```

Эта функция принимает на вход указатели на 2 числа.
Она возвращает 0, если числа равны.
Она возвращает положительное число, если первое число больше второго.
Она возвращает отрицательное число, если второе число больше первого.

Пример вызова функции сравнения:

```
     int a = 2;
     int b = 3;
     if (Compare (&a, &b) < 0) {
          printf ("a < b ");
     }
     if (Compare (&a, &b) <= 0) {
          printf ("a <= b ");
     }
     if (Compare (&a, &b) >= 0) {
          printf ("a >= b ");
     }
     if (Compare (&a, &b) > 0) {
          printf ("a > b");
     }
```

Добейтесь корректной работы программы.
Можно ли использовать такую функцию сравнения для целых чисел в диапазоне от -2 000 000 000 до 2 000 000 000? Если нет, то почему? Как эту функцию можно скорректировать, чтобы это стало возможным?

3. Везде где возможно замените вызов функции Compare вызовом функции по указателю на функцию.

Сам указатель требуется передавать в функцию сортировки и дальше в функцию поиска минимального элемента из основной процедуры.
Для сокращенной записи объявления переменных типа «указатель на функцию сравнения» используйте конструкцию typedef.

```
typedef int (*CompareFunctionType) (void *, void *);
```
Теперь переменную это типа можно объявить так:
```
CompareFunctionType compare_function_pointer = Compare;
```
Что равнозначно объявлению переменной:
```
int (*compare_function_pointer) (void *, void *) = Compare;
```
Но первый путь короче и нагляднее.
Тогда заголовки функций сортировки и поиска минимального элемента изменятся следующим образом:
```
int FindMinimumIndex (int input_data[], int length,
                      int first_index, int last_index,
              CompareFunctionType compare_function_pointer);

void SelectionSort (int input_data[], int length,
              CompareFunctionType compare_function_pointer);
```

Вызовы функции сравнения будут выглядеть так:
```
if ((*compare_function_pointer) (&a, &b) < 0) {
     printf ("a < b ");
}
```
А вызов сортировки будет выглядеть так:
```
int Compare (void *first_pointer, void *second_pointer);
int my_int_array[10];
int length = 10;
SelectionSort (my_int_array, length, Compare);
```
Добейтесь корректной работы программы.

4. Объявите две различные функции сравнения:
```
int CompareInt0to9 (void *first_pointer, void *second_pointer);
int CompareInt9to0 (void *first_pointer, void *second_pointer);
```

Сделайте это так, чтобы при использовании первой функции массив сортировался по возрастанию, а при использовании второй - по убыванию. Отлаженный в третьем пункте алгоритм сортировки и поиска минимума тут менять запрещено!

Объявите в основной программе массив из функций типа CompareFunctionType.
```
const int N = 2;
CompareFunctionType compareFunctionPointers[ N ];
```
Инициализируйте его элементы указателями на функции CompareInt0to9 и CompareInt9to0.
Пусть в зависимости от желания пользователя в функцию сортировки массива передается либо одна, либо другая функция сравнения.

5. Требуется написать универсальную функцию, которая может сортировать массив независимо от типа его элементов.

Пусть теперь функция сортировки принимает на вход просто указатель на непрерывную область памяти input_data, количество элементов в массиве length, длину одного элемента массива в байтах width и указатель на функцию сравнения элементов compareFunctionPointer.

```
typedef int (*CompareFunctionType) (void *, void *);

int FindMinimumIndex (void * data_array, int length, int width,
                      CompareFunctionType compare_function_pointer,
                             int first_index, int last_index);

void Swap (void *first_pointer, void *second_pointer, int width);
/*Указание: напишите отдельно функцию SwapChar, меняющую
местами содержимое однобайтовых ячеек памяти, и вызовите ее
width раз из функции Swap.*/
void SwapChar (char *first_pointer, char *second_pointer);

void SelectionSort (void *data_array, int length, int width,
                    CompareFunctionType compare_function_pointer);
```

Напишите имплементацию этих функций так, чтобы с их помощью можно было сортировать по возрастанию разные массивы следующим образом:

```
//прототипы функций сравнения
int CompareCharAtoZ (void *first_pointer, void *second_pointer);
int CompareInt0to9 (void *first_pointer, void *second_pointer);
//объявления массивов
const int length = 10;
int my_int_array [length];
char my_char_array[100];
//Заполнение массивов пропущено…
//Вызов функции сортировки
SelectionSort ((void *)my_int_array, length, 4,
                                              CompareInt0to9);
SelectionSort ((void *)my_char_array, 100,
                  sizeof (my_char_array [0]), CompareCharAtoZ);
```

Если самостоятельно написать код не получается, посмотрите на вариант на следующей странице.

Протестируйте программу, добейтесь ее корректной работы и сохраните код для практикума #15.

Приложение А. Возможный вариант имплементации FindMinimumIndex и Swap.

```
void SwapChar (char *first_pointer, char *second_pointer) {
     char temp = *first_pointer;
     *first_pointer = *second_pointer;
     *second_pointer = temp;
     return;
}

void Swap (void *first_pointer, void *second_pointer,
                                                    int width) {
     for (int byte_number = 0; byte_number < width;
                                                  ++byte_number) {
          SwapChar ((char *)first_pointer + byte_number,
                         (char *)second_pointer + byte_number);
     }
     return;
}

int FindMinimumIndex (void *data_array, int length, int width,
                CompareFunctionType compare_function_pointer,
                int first_index, int last_index) {
     char *bytevise_array = (char *)data_array;

     int minimum_index = first_index;
     int minimum_byte_number = minimum_index * width;
     int current_byte_number = 0;

     for (int element_number = first_index;
               element_number <= last_index; ++element_number) {

          current_byte_number = element_number * width;
          if ((*compare_function_pointer) (
                  &bytevise_array [current_byte_number],
                  &bytevise_array [minimum_byte_number]) < 0 ) {
               minimum_byte_number = current_byte_number;
          }
     }

     minimum_index = minimum_byte_number / width;
     return minimum_index;
}
```