## Задача ku03-5: kr03-5 (upsolving)

Напишите функцию сортировки массива целых чисел в стиле Continuation Passing Style.
Особенность стиля CPS в том, что функции не возвращают свой результат. Вместо этого они передают свой результат следующей функции, которая передается в качестве аргумента. **После этого управление не возвращается в функцию.**
Предположим для простоты, что функция, аналогичная функции `qsort_r`, поддерживает сортировку целых чисел. Функция сравнения двух элементов массива может выглядеть следующим образом:

```
int sort_func(int v1, int v2, void *cntx)
{
    return v1 - v2;
}
```

(в упрощенном предположении, что разность v1 - v2 всегда представима в int). Сама функция сортировки тогда может быть такой:

```
void sort_ints(int *arr, int size, int (*func)(int, int, void *),
void *cntx)
{
    // эта функция вызывает func(v1, v2, cntx) для сравнения двух
значений
}
```

Параметр `cntx` — это *контекст*, с помощью которого функция, вызывающая `sort_ints`, может передать дополнительную информацию в функцию сравнения двух элементов.
В **стиле с продолжениями** (continuations) функция сравнения двух элементов будет следующей:

```
void sort_func_cc(int v1, int v2, void (*cc)(int result, void
*cntx), void *cntx)
{
    cc(v1 - v2, cntx);
    // never get here
    abort();
}
```

То есть функция сравнения больше не возвращает результат сравнения, вместо этого вызывается указанная функция `cc`, которой передается результат сравнения `result` и контекст `cntx`, который сама функция `sort_func_cc` получила на вход.
Итак, **напишите функцию `sort_with_cc`** со следующим прототипом:

```
void
sort_with_cc(
        int *data, // сортируемый массив элементов
        int size,  // количество сортируемых элементов
        void (*cmp)(    // функция сравнения двух элементов с
продолжением
                int v1,
                int v2,
```

```
                void (*sort_cc)(int result, void *sort_cntx),
                void *sort_cntx),
        void (*cc)(     // функция, которая будет вызвана после
завершения сортировки
                int *sorted_data,  // отсортированный массив
                int sorted_size,   // количество элементов
                void *user_cntx),  // контекст, полученный на вход
sort_with_cc
        void *user_cntx);
```

В `sorted_data` можно передавать тот же самый указатель `data` (не нужно делать копию массива). Сортировка может работать квадратичное время.

Например, следующая программа считывает массив целых чисел заданного размера и сортирует его по возрастанию (считается, что переполнения невозможны).

```
void simple_compare_cc(int v1, int v2, void (*cc)(int result, void
*cntx), void *cntx)
{
    cc(v1 - v2, cntx);
    abort();
}

void  simple_print_cc(int  *sorted_data,  int  sorted_size,  void
*cntx)
{
    printf("%s\n", (char*) cntx);
    for (int i = 0; i < sorted_size; ++i) {
        printf("%d\n", sorted_data[i]);
    }
    exit(0);
}

int main()
{
    int n;
    scanf("%d", &n);
    int *arr = calloc(n, sizeof(*arr));
    for (int i = 0; i < n; ++i) {
        scanf("%d", &arr[i]);
    }
    sort_with_cc(arr,   n,   simple_compare_cc,   simple_print_cc,
"100500");
    // should never get here
    abort();
}
```

Пример входных данных:

```
5
5 4 3 2 1
```

Результат работы:

```
100500
1
2
3
4
5
```