Лабораторная работа 2
Динамическая память.

Помимо статической и стековой памяти, существует еще практически неограниченный ресурс памяти, которая называется *динамическая*, или *куча* (*heap*). Программа может захватывать участки динамической памяти нужного размера. После использования ранее захваченный участок динамической памяти следует освободить.

На уровне библиотек в Си создан механизм порождения и уничтожения переменных самой работающей программой. Такие переменные называются динамическими, а область памяти, в которой они создаются – динамической памятью или «кучей». Куча организуется в одном или нескольких дополнительных сегментах памяти, выделяется программе операционной системой. Все в целом называется также системой динамического распределения памяти (ДРП). Перечислим основные свойства динамических переменных:

- динамические переменные создаются и уничтожаются работающей программой путем выполнения специальных операторов или вызовов функций;
- количество и размерность динамических переменных (массивов) может меняться в процессе работы программы. Это определяется числом вызовов соответствующих функций их параметрами;
- динамическая переменная не имеет имени, доступ к ней возможен только через указатель;
- функция создания динамической переменной ищет в «куче» свободную память необходимого размера и возвращает указатель на нее (адрес);
- функция уничтожения динамической переменной получает указатель на уничтожаемую переменную.

Наиболее важным свойством динамической переменной является ее «безымянность» и доступность по указателю. Таким образом, динамическая переменная не может быть доступна «сама по себе», а только опосредованно через другие переменные или обычные именованные указатели. Динамическая переменная может, в свою очередь, содержать один или несколько указателей на другие динамические переменные. В этом случае мы получаем динамические структуры данных, в которых количество переменных и связи между ними могут меняться в процессе работы программы (списки, деревья);

Работа с динамическими переменными и системой ДРП имеет ряд особенностей и сложностей. Они усугубляются еще и тем, что в Си в соответствии с требованиями эффективности программного кода, функции библиотеки минимально защищены от ошибок программирования:

Для работы с динамической памятью в языке Си используются следующие функции:

- malloc,
- calloc,
- free,
- realloc.

В языке Си для захвата и освобождения динамической памяти применяются стандартные функции malloc и free, описания их прототипов содержатся в стандартном заголовочном файле "stdlib.h".

```
void* malloc(size_t size);
```

здесь `size` - это размер захватываемого участка в байтах, `size_t` - имя одного из целочисленных типов, определяющих максимальный размер захватываемого участка.

Функция `malloc` возвращает адрес захваченного участка памяти или ноль в случае неудачи (когда нет свободного участка достаточно большого размера). Функция `free` освобождает участок памяти с заданным адресом. Для задания адреса используется указатель общего типа `void*`. Адрес выделенной области памяти также возвращается в виде указателя типа `void*` - абстрактный адрес памяти без определения адресуемого типа данных.

После окончания работы с выделенной динамически памятью нужно освободить ее. Для этой цели используется функция free, которая возвращает память под управление ОС:

```
void free(void *p);
```

В качестве входного параметра в `free` нужно передать указатель, значение которого получено из функции `malloc`.

Вызов free на указателях полученных не из malloc (например, free(p+10)) приведет к неопределенному поведению. Это связанно с тем, что при выделении памяти при помощи malloc в ячейки перед той, на которую указывает возвращаемый функцией указатель операционная система записывает служебную информацию. При вызове free(p+10) информация находящаяся перед ячейкой (p+10) будет трактоваться как служебная.

```
void* calloc(size_t nmemb, size_t size);
```

Функция работает аналогично `malloc`, но отличается синтаксисом (вместо размера выделяемой памяти нужно задать количество элементов и размер одного элемента) и тем, что выделенная память будет обнулена.

Например, после выполнения

```
int * q = (int *) calloc(1000000, sizeof(int))
```

q будет указывать на начало массива из миллиона int`ов инициализированных нулями.

Пример:

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

int main() {
    int n;
    int *a;

    printf("Введите число элементов: ");
    scanf("%d", &n);
    if (n <= 0)
        exit(1);
    // Захватываем память под массив простых чисел
    a = (int *) malloc(n * sizeof(int));

    a[0] = 2; k = 1;      // Добавляем двойку в массив
    printf("%d ", a[0]); // и печатаем ее
    free(a);
}
```

Перераспределение памяти

```
void *realloc(void *ptr, size_t size);
```

Функция изменяет размер выделенной памяти (на которую указывает `ptr`, полученный из вызова `malloc`, `calloc` или `realloc`). Если размер указанный в параметре `size` больше, чем тот, который был выделен под указатель `ptr`, то проверяется, есть ли возможность выделить недостающие ячейки памяти подряд с уже выделенными. Если места недостаточно, то выделяется новый участок памяти размером `size` и данные по указателю `ptr` копируются в начало нового участка.

Утечка памяти (Memory leak)

Если процесс попросил у ОС память, а затем про нее забыл и более не использует, это называется утечкой памяти.

Утечки памяти не являются критической ошибкой и в небольшом масштабе допустимы, если процесс работает очень недолго (секунды). Однако при разработке сколько-нибудь масштабируемого и выполняющегося продолжительное время приложения, допущение даже маленьких утечек памяти — серьезная ошибка.

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

```
void swap_arrays(int *A, int *B, size_t N)
{
    int * tmp = malloc(sizeof(int)*N);
    for(size_t i = 0; i < N; i++)
        tmp [i] = A[i];
    for(size_t i = 0; i < N; i++)
        A[i] = B[i];
    for(size_t i = 0; i < N; i++)
        B[i] = tmp [i];
}

int main()
{
    int A[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int B[10] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
    swap_arrays(A, B, 10);

    int *p;
    for(int i = 0; i < 10; i++) {
        p = malloc(sizeof(int));
        *p = 0;
    }
    free(p);

}
```

Задания на ЛБ.

Общая структура заданий:

1. С клавиатуры пользователем вводится число элементов массива.
2. Выделяется память заданного размера.
3. Выделенная память заполняется случайными числами.
4. Выводится на экран заполненный массив.
5. Выполняется обработка массива в соответствии с заданием.
6. В случае необходимости изменения его размера выполняется перераспределение памяти.
7. Выводится на экран содержимое измененного в результате работы программы массива.
8. Освобождается память и завершается работа программы

В процессе работы программы необходимо выводить на экран вспомогательную информацию: размер выделенной или перераспределенной памяти, адрес ее начала.

Задача 1.

1. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-40,30]. Удалить из него все элементы, которые состоят из одинаковых цифр (включая однозначные числа).

2. Вставить число *K* перед всеми элементами, в которых есть цифра 1.

3. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-10,60]. Удалить из него все элементы, в которых последняя цифра четная, а само число делится на нее.

4. Вставить элемент со значением *K* до и после всех элементов, заканчивающихся на цифру *K*.

5. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-35,75]. Удалить из него все элементы, первая цифра которых четная.

6. Вставить число *K1* после всех элементов, больших заданного числа, а число *K2* – перед всеми элементами, кратными трем.

7. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-45,95]. Удалить из него все элементы кратные 7 и принадлежащие промежутку [*a, b*].

8. Вставить число *K* между всеми соседними элементами, которые имеют одинаковые знаки.

9. Переставить в обратном порядке часть массива между элементами с номерами *K1* и *K2*, включая их.

10. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-20,50]. Удалить из него все элементы, в записи которых есть цифра 5.

12. Поменять местами первый положительный и последний отрицательный элементы.

13. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-40,30]. Удалить из него все четные элементы, у которых последняя цифра 2.

14. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-10,60]. Удалить из него все элементы, в которых последняя цифра четная, а само число делится на нее.

15. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-20,50]. Удалить из него все элементы, первая цифра которых четная.

16. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [-30,30]. Удалить из него все элементы кратные 7 и принадлежащие промежутку [*a, b*].

17. Дан массив целых чисел из *n* элементов, заполненный случайным образом числами из промежутка [10,100]. Удалить из него все элементы, в записи которых есть цифра 3.

18. Вставить число *K* перед всеми элементами, в которых есть цифра 2.

Задача 2.

1. Вставить максимальный элемент массива после всех элементов, в которых есть цифра 1.

2. Переставить первые k и последние k элементов местами, сохраняя порядок их следования.
3. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-10,60]. Удалить из него все элементы, в которых последняя цифра нечетная, а само число кратно 3.
4. Вставить элемент со значением *K* после всех четных элементов, начинающихся на цифру *K*.
5. Вставить число *K1* после всех элементов, больших заданного числа, а число *K2* – после всех элементов, кратных пяти.
6. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-135,175]. Удалить из него все элементы, первая и последняя цифра которых четная.
7. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-95,95]. Удалить из него все отрицательные элементы кратные 5 и принадлежащие промежутку $[a,b]$.
8. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-50,50]. Удалить из него все элементы, в записи которых последняя цифра равна 0.
9. Вставить значение минимального элемента массива после всех четных элементов.
10. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-140,140]. Удалить из него все элементы, у которых первая и вторая цифры одинаковые.
11. Переставить первые два и средние два элемента местами, сохраняя порядок их следования (количество элементов – четное).
12. Вставить максимальное значение элементов массива перед всеми элементами, в записи которых есть цифра 1.
13. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-200,500]. Удалить из него все элементы, в записи которых есть цифра 0.
14. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-20,50]. Удалить из него все элементы, в которых последняя цифра нечетная, а само число кратно 2.
15. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [100, 275]. Удалить из него все элементы, первая и последняя цифра которых нечетная.
16. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-65,65]. Удалить из него все отрицательные элементы кратные 3 и принадлежащие промежутку $[a,b]$.
17. Дан массив целых чисел из n элементов, заполненный случайным образом числами из промежутка [-100,100]. Удалить из него все элементы, у которых первая и вторая цифры разные.
18. Вставить значение минимального элемента массива после всех нечетных элементов.