Лекция IV

25 октября 2017

Со строками ещё не закончили

Работа с текстом

Преобразование строк в числа с помощью библиотеки **<stdlib.h>**

```
(1) int       atoi( str )
(2) double    atof( str )
(3) long      atol( str )
(4) long long atoll( str )
```

Данные четыре функции пытаются преобразовать переданную им строку в соответствующее числовое значение. Если преобразование не удалось, то возвращаемый результат не определён (UB).

Преобразование строк в числа с помощью функций из **<stdlib.h>**

```
1  char sf[] = "456.7788",
2       si[] = "-7485",
3       sl[] = "313377317135";
4
5  double    d_num = atof(  sf );
6  int       i_num = atoi(  si );
7  long long l_num = atoll( sl );
8
9  printf("Действительное число: %f\n", d_num);
10 printf("Целое число: %d\n", i_num);
11 printf("Длинное целое: %lld\n", l_num);
```

Преобразование строк в числа

```
1 double d_val = atof("1.454sdfdf");
2 printf("Так всё хорошо: %f\n", d_val);
3 // Напечатает 1.454
4
5 int i_val = atoi("sdadd124");
6 printf("А так - не очень: %d\n", i_val);
7 // Непонятно, что за число напечатает
```

Работа с текстом. C++ строки

Преобразование чисел в строку (<**stdio.h**>)

```
int sprintf(char *str, const char *format, ...)
```

- Функция делает тоже самое, что и **printf**, но получившийся текст не выводит в терминал, а сохраняет в строку **str**
- Число аргументов после двух обязательных является переменным (зависит от спецификаторов в аргументе **format**
- **str** - строка, куда сохраняется результат
- Функция возращает длину получившейся строки, если её работа прошла успешно. Иначе - возращается отрицательное значение
- **sprintf** никак не контролирует длину строки **str**

Преобразование чисел в строку: пример **sprintf**

```
1 double val1 = 3.4, val2 = 5.5;
2 char buffer[50];
3 char format[] = "%.3f плюс %.3f равно %.3f";
4
5 int buf_len = sprintf(buffer, format,
6         val1, val2, val1 + val2);
7 printf("Строка <<%s>> имеет длину %d",
8        buffer, buf_len);
```

Как ещё форматировать числа

можно посмотреть здесь:
https://github.com/posgen/OmsuMaterials/wiki/Format-output-in-C

Работа с текстом. C++ строки

Преобразование чисел в строку (<**stdio.h**>)

```
int snprintf(char *str, size_t count,
             const char *format, ...)
```

- Работает аналогично **sprintf**, но в строку **str** записывает не более, чем **count - 1** символов. Последним ставится *символ окончания строки*

```
1 double val = 14.281246846842168412862441 31;
2 char buffer[20];
3 char format[] = "Число: %.15f";
4
5 snprintf(buffer, 20, format, val);
6 printf("Полученная строка: %s", buffer);
```

Указатели и динамическое управление памятью

Указатели и динамическая память

Что входит в понятие **динамическое управление памятью**?

- Стандартная библиотека C предоставляет функции для получения от ОС блоков памяти, заданного размера
- Размер задаётся в **байтах**
- Динамический блок памяти существует до тех пор, пока не будет вызвана функция для его возращения в ОС
- Доступ к таким блокам осуществляется только при помощи указателей

Возникает логичный вопрос: зачем?

Указатели и динамическая память

На первый взгляд, есть же массивы переменной длины

```
1 size_t dyn_size;
2 printf("Введите размер массива: ");
3 scanf("%lu", &dyn_size);
4
5 double my_array[dyn_size];
6 // Работаем с my_array
```

Что будет, если хотим вернуть массив из функции?

Попытка №1

```
1  int* make_magic_array(const size_t sz)
2  {
3    int vec[sz];
4    for (size_t i = 0; i < sz; ++i) {
5      vec[i] = rand() % sz;
6    }
7
8    return vec;
9  }
10
11 // НЕ ПОВТОРЯТЬ: получили адрес памяти, которая
12 // была возвращена ОС после вызова функции
13 int *p_arr = make_magic_array(10);
14 printf("Второе число: %d\n", p_arr[1]);
```

Ограничение на массивы: массивы языка C **не могут быть использованы** в качестве возвращаемых значений из функций.

Указатели и динамическая память

Управление динамической памятью полезно, так как

- в различных задачах нужны блоки памяти под переменные или массивы, время жизни которых должно превышать локальную область видимости (пример - очереди, стеки, списки). В задачах вычислительной физики - особенно;
- на некоторых ЭВМ максимальный размер массива, который можно получить с помощью динамической памяти, превышает таковой для статических массивов.

Указатели и динамическая память

Функции для получения блока динамической памяти заданного размера

```
(1) void* malloc( size_t size_in_bytes );
(2) void* calloc( size_t count_of_blocks,
                  size_t size_of_one_block );
```

- **(1)** - функция запрашивает у ОС блок памяти, размером **size_in_bytes** байт, и возвращает указатель на начало блока (адрес первого байта)
- **(2)** - запрашивает блок памяти, размером **count_of_blocks** x **size_of_one_block** байт и устанавливает **все биты блока** равными нулю
- Обе функции возвращают значение **NULL**, если выделение памяти прошло неудачно
- Поскольку возвращаемый тип - указатель на **void**, для работы с полученной памятью требуется явное приведение к конкретному типу

Указатели и динамическая память

Функция для возращения блока динамической памяти в ОС

```
void free( void *ptr );
```

- Память, на которую указывает **ptr** должна быть выделена с помощью **malloc** или **calloc**!
- Иначе - что случится в этой функции, стандартом языка не определено (UB)
- Однако, если в переданном в функцию указателе содержится значение **NULL**, то функция ничего не делает

Пример использования **malloc**

```
1  int *p1;
2  // Выделили память под 1 элемент типа int
3  p1 = (int*) malloc(1 * sizeof(int));
4  if (p1 != NULL) {
5    p1[0] = 19;
6    printf("Первый элемент p1: %d", *p1);
7  }
8
9  size_t arr_sz = 8;
10 int *p2 = (int*) malloc(arr_sz * sizeof(int));
11 if (p2 != NULL) {
12   for (size_t i = 0; i < arr_sz; ++i) {
13     p2[i] = -10 + rand() % 55;
14     printf("p2[%lu] = %d\n", i, p2[i]);
15   }
16 }
17 // Никогда не забывать
18 free(p1); free(p2);
```

Пример использования **calloc**

```
1  size_t my_sz;
2
3  printf("Введите размер массива: ");
4  scanf("%lu", &my_sz);
5
6  double *p_arr = (double*) calloc(my_sz,
7                                   sizeof(double));
8
9  if (p_arr != NULL) {
10   for (size_t i = 0; i < my_sz; ++i) {
11     // Каждый элемент будет равен 0.0
12     printf("arr[%lu] = %f\n", i, p_arr[i]);
13   }
14 }
15
16 free(p_arr);
```

Пример - возврат массива из функции

```
1  int* make_magic_array(size_t sz)
2  {
3    int *pi_arr = (int*) calloc(sz, sizeof(int));
4    if (pi_arr == NULL) { return pi_arr; }
5
6    for (size_t i = 0; i < sz; ++i) {
7      for (size_t j = 0; j < 9; ++j) {
8        pi_arr[i] += rand(); }
9      pi_arr[i] /= 9;
10   }
11
12   return pi_arr;
13 }
14
15 int *my_arr = make_magic_number(30);
16 if (my_arr != NULL) {  /*действия с my_arr*/ }
17
18 free(NULL);
```

Указатели и динамическая память

Пример - как создать утечку памяти

```
1  double get_complex_average(size_t count)
2  {
3    if (count == 0) {
4      return 0.0;
5    }
6
7    double *arr, average;
8    arr =  = (double*) malloc(count * sizeof(↩
         double));
9    // Сложная обработка массива
10   // И ни одного вызова free
11   return average;
12 }
13
14 get_complex_average(10);
15 get_complex_average(1500);
```

Пример - как устранить утечку памяти

```
1  double get_complex_average(size_t count)
2  {
3    if (count == 0) {
4      return 0.0;
5    }
6
7    double *arr, average;
8    arr = (double*) malloc(count * sizeof(↩
         double));
9    // Сложная обработка массива
10   free(arr); // Не забываем
11
12   return average;
13 }
14
15 get_complex_average(10000);
```

Пример - многомерные указатели

```
1  int **matrix;
2  size_t rows, cols;
3
4  printf("|Введите размерность матрицы");
5  scanf("%lu, %lu", &rows, &cols);
6
7  // Выделяем массив, элементы которого — указатели
8  matrix = (int **) malloc(rows * sizeof(int*));
9
10 // Берём каждый элемент, созданного массива...
11 for (size_t i = 0; i < rows; ++i) {
12   // .. и создаём для него уже массив элементов
13   // типа int
14   matrix[i] = (int *) malloc(cols * sizeof(int))↩
         ;;
15 }
16
17 //... см. следующий слайд
```

Пример - многомерные указатели

```
1  //... см. предыдущий слайд
2  for (size_t i = 0; i < rows; ++i) {
3    for (size_t j = 0; j < cols; ++j) {
4      matrix[i][j] = i + j;
5    }
6  }
7
8  // Какая-то работа с matrix
9
10 // Возращение динамической памяти в ОС
11 for (size_t i = 0; i < rows; ++i) {
12   // Удаляем каждую строку
13   free(matrix[i]);
14 }
15 // Удаляем сам массив строк
16 free(matrix);
```