

# Функции в С

Семестр 1

Семинар 3

# Переменный размер массива

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

```
int main()
```

```
{
```

```
    int n;
```

```
    scanf ("%d", &n);
```

```
    int * p = (int *) malloc (n * sizeof (int) );
```

```
        // выделили память размера n * sizeof(int) байт
```

```
        //p - указатель на начало выделенной памяти
```

```
    p[3] = 10;      *(p+5) = 20;    scanf ("%d", p+2);
```

```
    printf("sum = %d\n", p[3] + p[5] + p[2]);
```

```
    free (p);      // освобождаем память
```

```
}
```

# Задачи

0. Выяснить, является ли строка палиндромом (А роза упала на лапу Азора).
  - чистый палиндром («qwertyytrewq»)
  - игнорируя пробелы («q wer ty ytre wq»)
1. Сортировка одномерного массива целых чисел.
  - методом простого выбора
  - пузырьком
2. Перевернуть каждое второе слово в строке.
3. Считать слово с экрана (до пробела). Удалить все повторяющиеся буквы. Вывести остальные на экран
4. Считать с экрана длину массива и сам массив целых чисел. Вывести вначале все четные, отсортированные по возрастанию, потом все нечетные, отсортированные по убыванию.
5. Умножение матриц  $N \times M$  и  $M \times L$ , содержащих целые числа.

# Применение функций

- обособленный участок кода, исполнение которого можно вызывать из любой точки кода
- программа == много маленьких функций  
(возможно, в разных файлах)
- + повторное использование кода
- + скрывает несущественные для других частей программы детали реализации
- + одна функция = одна задача (а не куча)
- + можно копировать отлаженный кусок кода из программы в программу

стандартная vs своя функция

- уже написана
- отлажена
- быстрые алгоритмы

# Объявление функции и ее вызов

тип\_результата имя\_функции (список параметров)

{

    декларация переменных  
    инструкции

    return переменная\_типа\_результата;

}

```
int main ( ) {  
    int x;  
    printf ("Hello, world\n");  
    x = 3*2;  
    return 0;  
}
```

# Объявление функции и ее вызов

```
int power (int x, int n) {  
    int res;  
    int i;  
    for (i=0, res=1; i<n; i++)  
        res *= x;  
    return res;  
}  
  
int main ( ) {  
    int y = power (3, 5);  
    return 0;  
}
```

Если функция не возвращает значения, return можно не писать

```
void hello () {  
    printf ("Hello\n");  
}
```

# Имя функции

- любой допустимый идентификатор, аналогично именам переменных:
  - 0-9 a-z A-Z \_
  - не может начинаться с цифры

- значимое

- + get\_font, set\_size, to\_upper, draw\_line

- + is\_empty, is\_not\_empty

- + SortArray, CleanCache

- skjfugig111, qwerty007, vachnadze531

не можете назвать – в функции слишком много разных действий, разбейте ее на несколько: готовит, стирает, моет полы

# Список параметров (аргументов)

- у каждого аргумента надо указать тип.
  - **правильно**: long power (int x, **int** n)
  - **ошибка**: long power (int x, n)
- функция *без параметров*
  - int getchar ( ) { ... }
  - int getchar ( **void** ) { ... }
- если параметров много, то функция решает слишком много задач



# О равноправии

Функция НЕ может быть описана  
внутри другой функции

```
int main ( ) {  
    ...  
    long power (int x, int n) {  
        ....  
    }  
    ...  
}
```

# Область видимости функции

видна только ниже объявления

(пока не объявили, пользоваться не можете)

```
int beta( ) {  
    ...  
    // компилятор не знает ничего об alpha  
    alpha(5);  
    ...  
}  
void alpha (int x) {  
    ...  
}
```

# Парадокс

Какая функция должна быть описана раньше?

```
void alpha ( ) {  
    beta ( );  
}  
void beta ( ) {  
    alpha ( );  
}
```

# Прототип функции (декларация)

тип\_результата имя\_функции  
(список *типов* параметров) ;

- тело функции описывается в произвольном месте
- long power (int x, int n); //  $x^n$
- long power (int, int);
  - достаточно компилятору
  - человеку не понятно

# Разрешаем парадокс

*сначала декларируем, потом  
используем*

```
void beta (void); // прототип
void alpha ( ) {
    beta ( );
}
void beta ( ) {
    alpha ( );
}
```

# Рекурсивный вызов функций

функция может вызвать саму себя

```
void numbers (int x) {  
    printf ("x=%d\n", x);  
    numbers (++x);  
}
```

- numbers (4); // что будет?
- главное – вовремя остановиться

# Задачи

0. Написать функцию сложения 2 целых чисел.
1. Написать функцию, находящую N-е число Фибоначчи.
2. Написать функцию, которая проверяет, является ли положительное число в десятичной записи палиндромом.
3. Написать программу, выводящую все палиндромы в десятичной записи от 0 до 1000000.
4. Написать функцию, считающую сумму геометрической последовательности.
5. Формула Валлиса расчета числа Пи. Аргумент функции – n. Посчитать значение Пи для n от единицы до 1000000, оценить минимальное n для достаточной точности.

$$\frac{\pi}{2} = \prod_{n=1}^{\infty} \frac{(2n)^2}{(2n-1)(2n+1)} = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdot \frac{8}{7} \cdot \frac{8}{9} \cdot \dots$$

# Характеристики переменных

- тип
- название
- значение
- адрес
- область видимости
  - от декларации и ниже
  - до конца блока, в котором определена

```
if ( y < 0 ) {  
    int x = 7;  
    ....  
}
```



# Глобальные переменные

- **определены вне всяких блоков и функций**
- **Область видимости** – от декларации до конца **файла**
  - пишут в начале файла
- **Время жизни** = время выполнения этой **программы**
- начальное значение = 0

```
int arr[5];  
void print_array () {  
    int i;  
    for (i=0; i<5; i++)  
        printf("%d ", arr[i]);  
}  
int main ( ) {  
    int i;  
    for (i=0; i<5; i++)  
        arr[i] = i;  
    print_array();  
    return 0;  
}
```

# Локальные переменные

- **Область видимости** – от декларации до конца функции (блока)
- **Время жизни** = время выполнения этой функции

Переменные `c` и `nl` **локальны** для функции `main`.

Переменная `check` **локальна** для цикла `while`.

```
int main () {  
    int c, nl;  
    while ( (c = getchar() ) != '\n' ) {  
        int check = 0;  
        if (c == '1') check = 1;  
        if (c == '2') check = 2;  
        nl += check;  
    }  
    printf ("%d \n", nl);  
    return 0;  
}
```

# Соккрытие переменных

(локальная переменная скрывает глобальную)

```
int x;
```

```
void incr (int x) {
```

```
    x++;
```

```
}
```

```
int main ( ) {
```

```
    int i, x=1;
```

```
    for (i=0; i < 3; i++) {
```

```
        incr (x);
```

```
        printf ("x=%d\n", x);
```

```
    }
```

```
    return 0;
```

```
}
```

# Аргументы функций

Тоже являются переменными. Можно менять их значение, использовать в арифметических и условных операторах.

- **Область видимости** – функция
- **Время жизни** = время выполнения этой функции
- **начальное значение** = значение аргумента при вызове

```
int power (int x, int n) {  
    int res;  
    int i;  
    for (i=0, res=1; i<n; i++)  
        res *= x;  
    return res;  
}  
  
int main ( ) {  
    int y = power (5, 2);  
    power (y-2, 4);  
    return 0;  
}
```

# Рекурсивный вызов функций

для каждого вызова функции создаются свои экземпляры локальных переменных и аргументов

```
void numbers (int x) {  
    int y = x + 1;  
    printf ("x=%d\n", x);  
    numbers (y);  
}
```

# Что будет выведено на печать?

```
void incr (int x) {  
    x++;  
}  
int main ( ) {  
    int i, x=1;  
    for (i=0; i < 3; i++) {  
        incr (x);  
        printf ("x=%d\n", x);  
    }  
    return 0;  
}
```

# Как менять значение аргументов?

- Перестроить алгоритм
- Глобальные переменные
  - небезопасная передача данных
  - неэффективное использование памяти
  - усложняет повторное использование кода
- Передача аргумента по ссылке

# Что будет выведено на печать?

```
int x;  
void incr ( ) {  
    x++;  
}  
int main ( ) {  
    int i;  
    for (i=0; i < 3; i++) {  
        incr ( );  
        printf ("x=%d\n", x);  
    }  
    return 0;  
}
```



# Передача в функцию адреса

```
void incr (int * x_pointer) {  
    (*x_pointer) ++;  
}
```

```
int main ( ) {  
    int i, x=1;  
    for (i=0; i < 3; i++) {  
        incr (&x);  
        printf ("x=%d\n", x);  
    }  
    return 0;  
}
```

# Передача в функцию массива

```
int arr[5];  
void print_array () {  
    int i;  
    for (i=0; i<5; i++)  
        printf("%d ", arr[i]);  
}  
int main ( ) {  
    int i;  
    for (i=0; i<5; i++)  
        arr[i] = i;  
    print_array();  
    return 0;  
}
```

# Передача в функцию массива

```
void print_array (int * _arr, int n) {  
    int i;  
    for (i=0; i<n; i++)  
        printf("%d ", _arr[i]); // _arr[i] == *(_arr+i)  
}  
  
int main ( ) {  
    int i, array[5] = {0, 1, 2, 3, 4}, arr_big[50] = {0};  
    print_array(array, 5); // array == &(amp;array[0])  
    print_array(arr_big, 50);  
    return 0;  
}
```

# Задачи

0. Написать функцию ввода массива с клавиатуры.
1. Написать программу, считывающую с экрана число  $n$  и  $n$  целых чисел. Потом вывод на экран вначале четных, потом нечетных.
2. Написать программу, считывающую с экрана число  $n$  и  $n$  целых чисел. Потом вывод на экран вначале простых, потом всех остальных. Использовать функцию из предыдущей части.
3. Написать функцию, которая выводит на экран массив с циклическим сдвигом на  $k$  элементов.
4. С экрана вводится строка символов. Необходимо выделить из нее две строки с сохранением порядка: состоящую только из цифр и состоящую только из заглавных букв латиницы. Каждую из двух проверить на палиндром и вывести на экран.