Функции в Си

Реализация структурного (модульного) подхода к разработке программ

Структурный подход: повторяющиеся фрагменты программ оформляются как подпрограммы (процедуры или функции). В основной программе – вызовы подпрограмм.

Программа на Си состоит из функций. Все функции записываются одна за другой.

При выполнении программы в первую очередь вызывается функция **`main()`**, все остальные функции должны вызываться из **`main()`** (или из функций, вызываемых из **`main()`**).

Программа:

Простейшая программа — только одна (главная) функция **`main()`**

Понятие и назначение функции

Функция - самостоятельная единица программы, спроектированная для решения конкретной задачи.

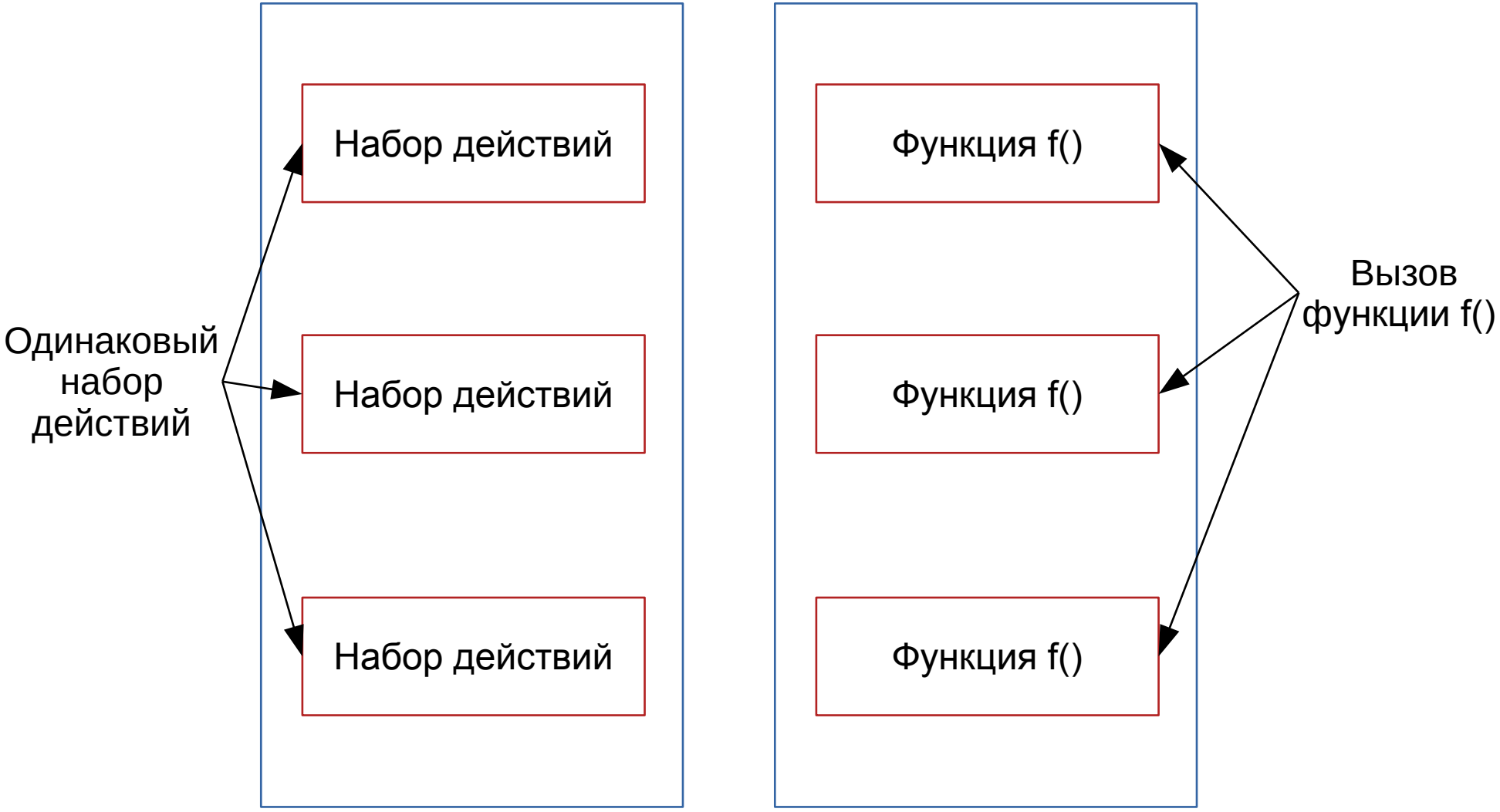
Задача, решаемая в функции, должны быть простой.

Функция в программе на Си включает в себя вызовы функций стандартной библиотеки, а также других пользовательских функций.

Назначение функций:

- Устранение повторяющихся фрагментов кода
- Многократное использование типовых фрагментов кода в разных программах
- Упрощение чтения кода и сопровождения программы.

Понятие и назначение функции



Описание функций

У функции должно быть имя и тип. *Имена пользовательских функций не должны совпадать с ключевыми словами и именами библиотечных функций.*

Функция может вернуть значение.

У функции могут быть аргументы (параметры).

`y = func22(x,z,w,q)` – вызов функции (обращение к функции)

Функция в Си может не возвращать значение. Тогда она должна иметь тип **`void`**.

Функция в Си может не иметь параметров или иметь переменное количество параметров (*вариадические функции*).

Если функция возвращает значение, то тип функции должен соответствовать типу возвращаемого значения.

Описание функций

Формальные и фактические параметры

```
<тип> <имя_функции>(<список формальных параметров>)
{
   <тело функции>
   return <возвращаемое значение>;
}

int main()
{
   …
   <переменная>=<имя_функции>(<список фактических параметров>)
   …
   return 0;
}
```

Параметры при описании и при вызове функции должны совпадать по позициям и по типам.

Описание функций

```
void show_menu()
{
   puts("1 – help");
   puts("2 – run");
   puts("3 – exit");
} /* не имеет параметров, не возвращает значений */

int qube(int b)
{
   return b*b*b;
}  /* имеет параметр-целое, возвращает целое значение */
```

Функция должна быть описана до ее вызова!

Описание функций

Пример lect-05-01.c

```
int qube(int b); /* Прототип функции. Компилятору этого
                    достаточно */

int main()
{
   …
   return 0;
}


int qube(int b)
{
   return b*b*b;
}  /* Реализация функции. Нужна редактору связей */
```

Описание функций

На ***этапе компиляции*** требуется иметь сведения о том, как обращаться к функции (наличие и тип возвращаемого значения, наличие, типы и порядок параметров).

На ***этапе сборки*** формируется исполняемый код, поэтому важно, что происходит в теле функций с их параметрами.

Оператор **`return`** передает управление в вызывающую (под)программу (функцию) и возвращает значение. Если значение не возвращается, управление передается после выполнения последнего оператора в теле функции.

Для функции **`main()`** вызывающей программой является операционная система. Функция **`main()`** возвращает код завершения программы (0 — корректное завершение, другие значения — коды ошибок, некорректное завершение).

Описание функций

```
int qube(int b);
/* ---------------- */
double discrim(double p, double q, double r)
{
   return q*q-4*p*r;
}
/* ----------------- */
double sphere_vol(int r)
{
   return 4*M_PI*qube(r)/3;
}
/* */
```

Фигурные скобки «тела» функции сносятся на следующую строку (заголовок функции имеет самостоятельное значение). Описания отделяются комментариями (пустыми или «-----»).

Описание функций

В «теле» функции могут быть описаны ***локальные переменные***, нужные для работы функции (например, параметры циклов).

Три варианта построения модуля (единицы трансляции) на Си:

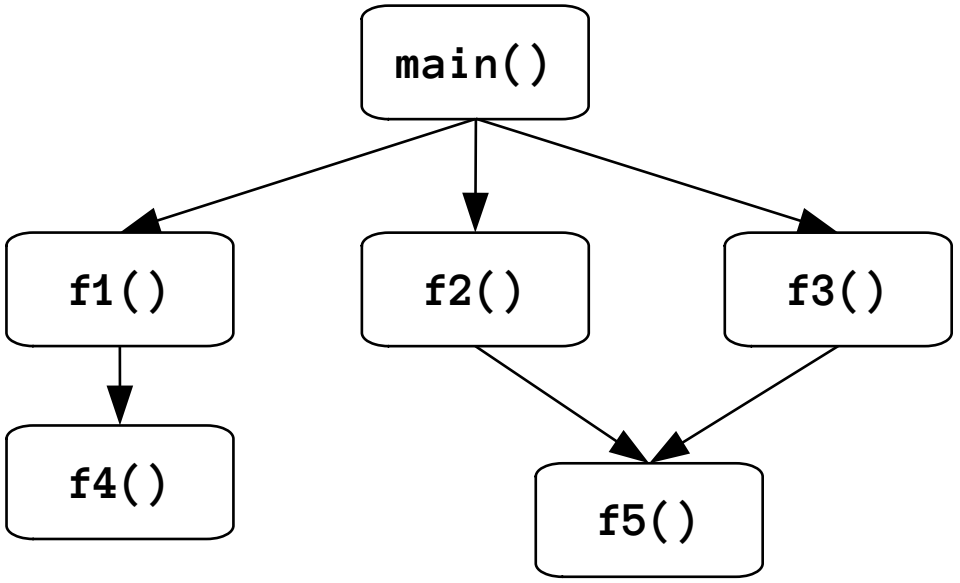
1. Прототипы функций → main() → Описания функций (в произвольном порядке).

Чтобы функция могла вызвать другую функцию, прототип вызываемой функции должен быть описан раньше прототипа вызывающей.

2. Описания функций (в порядке вызова) → main()

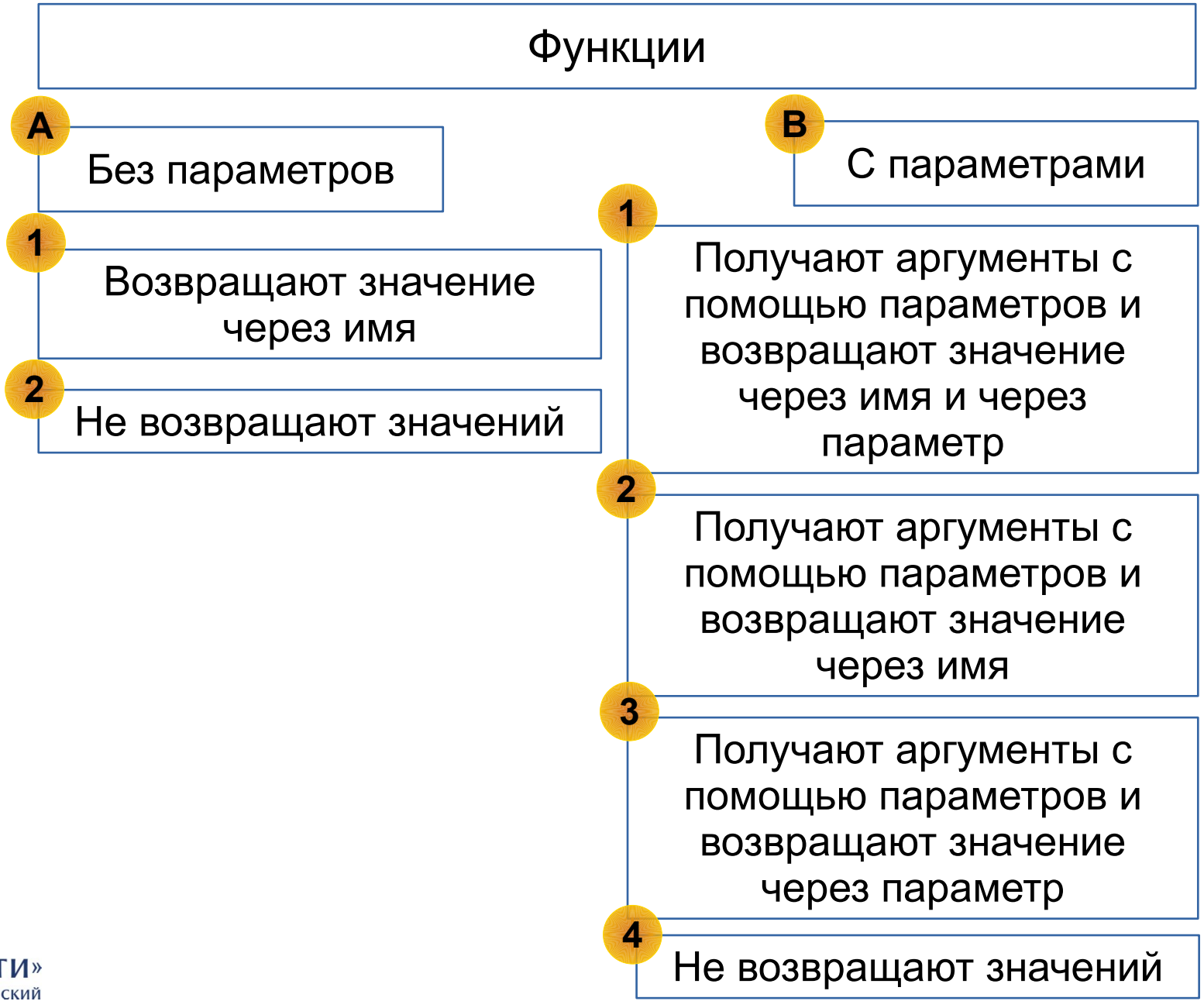
3. Прототипы и описания перемешаны (*плохая практика*).

Структура вызовов функций



Структура вызовов функций (иерархическая структура программы) — рисунок, показывающий взаимодействие функций: сколько функций в программе и как они между собой связаны.

Варианты функций



Оператор **`return`**

Оператор **`return`** завершает выполнение функции и передает в «точку вызова» значение выражения, записанного в функции после ключевого слова **`return`**. Значение передается через имя функции.

Синтаксически допустимо, что таких операторов в функции может быть несколько, они не обязательно должны быть в конце «тела» функции.

Подход структурного программирования — только один `return` !!!

Оператор `return`

```
int compare(int a, int b)
{
    if(a>b) return 0;
    else return 1;
} /* Плохая практика!!! */

int compare(int a, int b)
{
    if(a<0) exit(2); /* Завершение с кодом ошибки */
    if(a>b) return 0;
    else return 1;
} /* Плохая практика!!! */

int compare(int a, int b)
{
    int key;
    if(a<0)
    {
        key=-1;
        if(a>b) key=0;
    }
    else key=1;
    return key;
} /* Хорошая практика!!! */
        key=compare(m,n); /* Результат обрабатываем в main() */
```

Передача параметров

Передача по значению: содержимое аргумента копируется в формальный параметр функции. Изменения, сделанные в параметре, не влияют на значение переменной, используемой при вызове (что бы ни происходило с переменной внутри функции, в основной программе она имеет прежнее значение).

Пример lect-05-02.c

Передача по ссылке: в функцию копируется адрес аргумента. В теле функции этот адрес используется для доступа к значению аргумента, указанного при вызове. При этом изменения, сделанные в параметре функции, влияют на содержимое переменной, используемой при вызове.

Примеры: lect-05-03.c, lect-05-04.c

В Си обычно используется передача по значению для обычных переменных, а для указателей – передача по ссылке.

Указатель как параметр функции

Если параметр передается по ссылке (как указатель), то можно работать как со статической, так и с динамической памятью.

Для обработки массива с помощью функции всегда используется указатель (имя массива)!

Пример lect-05-05.c – нужно передавать и размер массива тоже!

Можно передать указатель на переменную, размещенную в динамической памяти — пример lect-05-06.c

При работе с массивами можно вернуть имя массива (как указатель) — примеры lect-05-07.c, lect-05-07a.c (вариант прототипов).

При использовании динамической памяти обязательно:
1. Проверить, что память выделилась (указатель — не NULL)
2. Очистить память после использования.

Функции и массивы

- Генерация массива N×M (для вещественных и для целых чисел)
- Перестановка элементов массива (**`swap()`**)
- Сортировка (для выбранного алгоритма, направление сортировки задается как дополнительный параметр)
- Ввод массива
- Вывод массива.

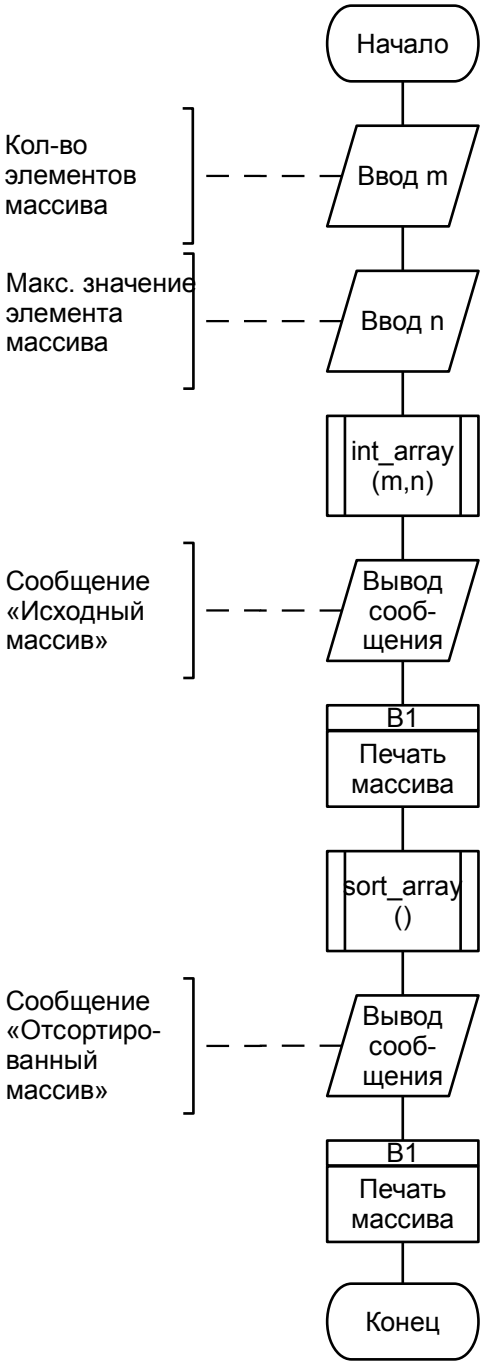
Вложенные функции

Если внутри пользовательской функции вызывается пользовательская функция с параметрами, то параметры нужно передать из вызывающей программы (`main()`) во внешнюю функцию.

Любая функция должна быть описана до использования.

Пример lect-07-08.c — используем функцию вызова генератора случайных чисел в диапазоне [0;N] в функции создания массива.

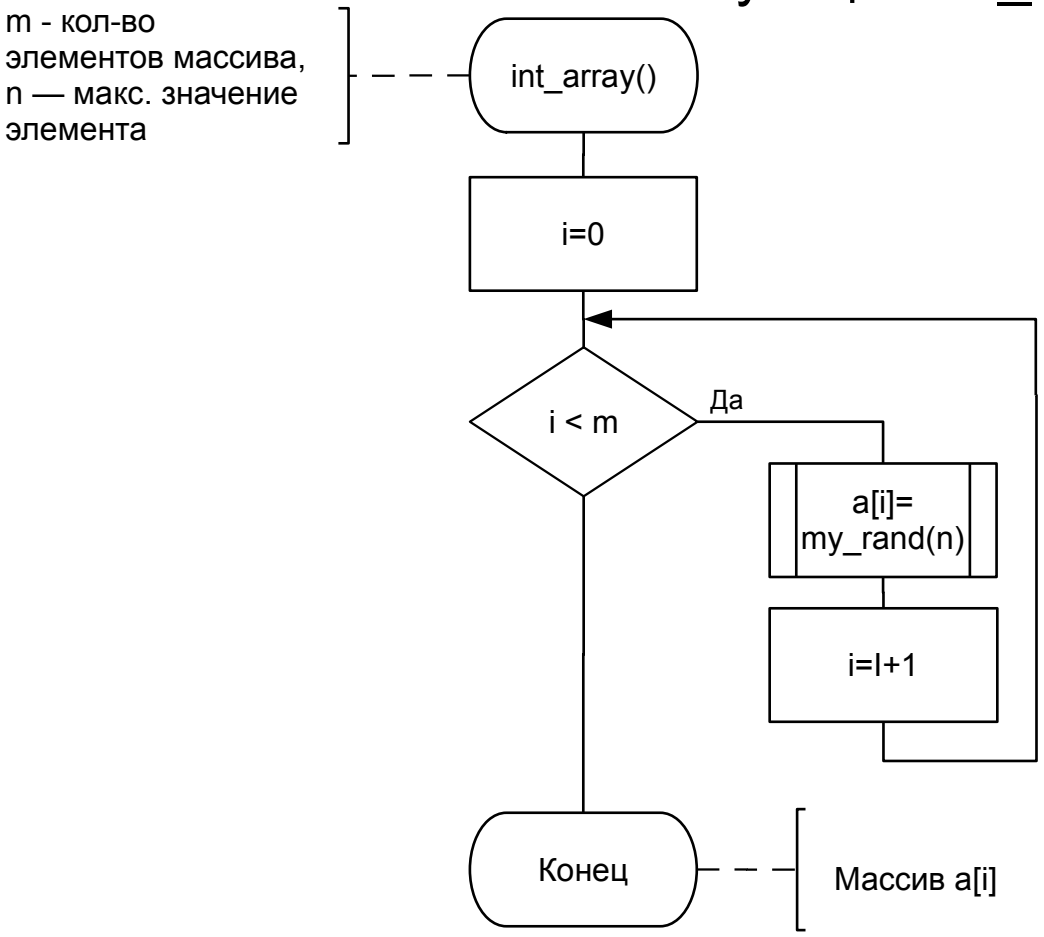
Функции на схемах алгоритмов

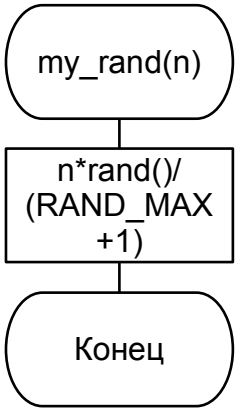


Основная программа по предыдущему примеру.

Функции на схемах алгоритмов

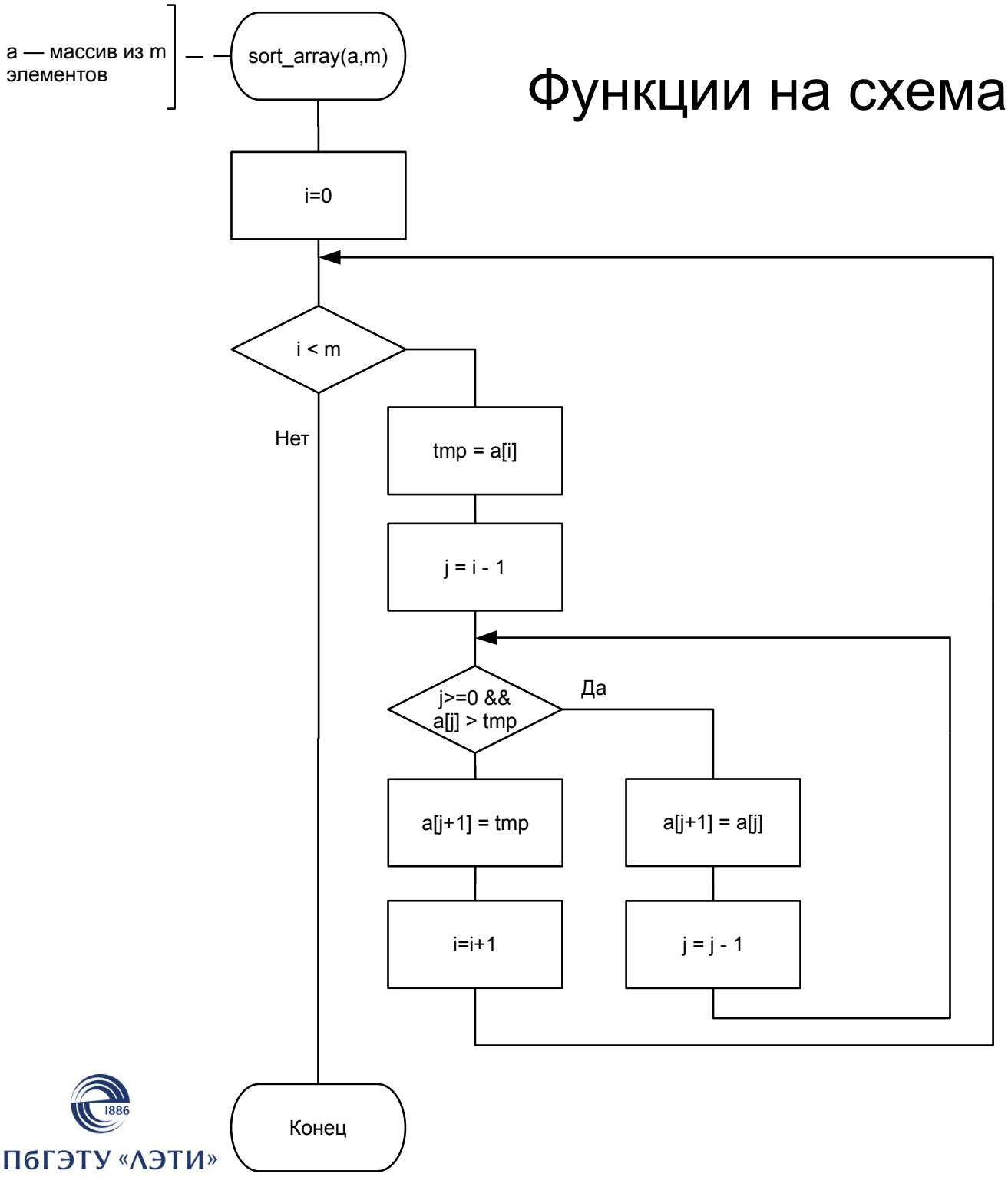
Функция int_array()



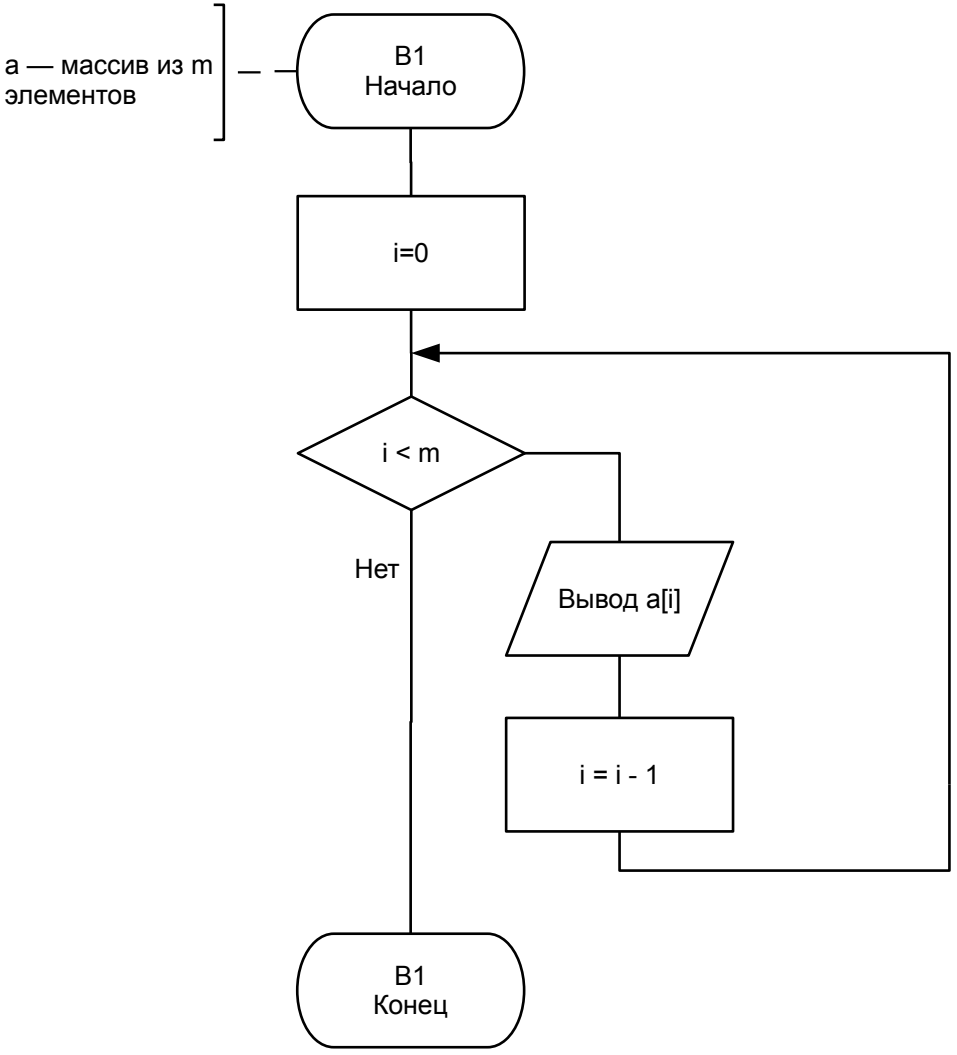


Функции на схемах алгоритмов

Функция sort_array()



Функции на схемах алгоритмов



Фрагмент программы, не являющийся функцией.

Задание: предложить и реализовать вариант этого фрагмента в виде функции.

Обработка символов

Задача: Написать функцию для определения, является ли символ, полученный с устройства ввода, символом-разделителем или нет. Символы-разделители определены в массиве.

Решение: пример lect-07-09.c.

Что осталось?

- Указатели на функции
- Рекурсивные функции
- Функции с переменным числом параметров (вариадические)
- Создание собственной библиотеки