Основы программирования на языке C

Темы занятий

- Введение в программирование
- Переменные и типы данных
- Базовые операции и приоритет операций
- Операции ветвления и цикла
- Ввод/Вывод информации
- Работа с массивами
- Работа с текстовыми и двоичными файлами
- Функции и директивы препроцессора
- Составные типы данных. Enum, структуры
- Работа с указателями. Выделение памяти
- Указатели и массивы. Указатель на функцию
- Связанные списки

Занятие 1. Темы

- Введение в программирование
- Трансляция компьютерной программы
- Инструменты разработчика ПО
- Понятие алгоритма
- Вывод информации на экран
- Переменные и типы данных
- Базовые операции и приоритет операций
- Оператор ветвления
- Операторы цикла
- Генерация псевдослучайных чисел
- Хранение целочисленных и вещественных типов данных

Введение в программирование

Программирование — процесс создания компьютерных программ.

Язык программирования — формальный язык, предназначенный для записи компьютерных программ. Язык программирования определяет набор лексических, синтаксических и семантических правил, определяющих внешний вид программы и действия, которые выполнит компьютер под её управлением.

Классификация языков программирования:

Низкоуровневые/Высокоуровневые

Интерпретируемые/Компилируемые

Императивные/Декларативные

Статической типизации/Динамической типизации

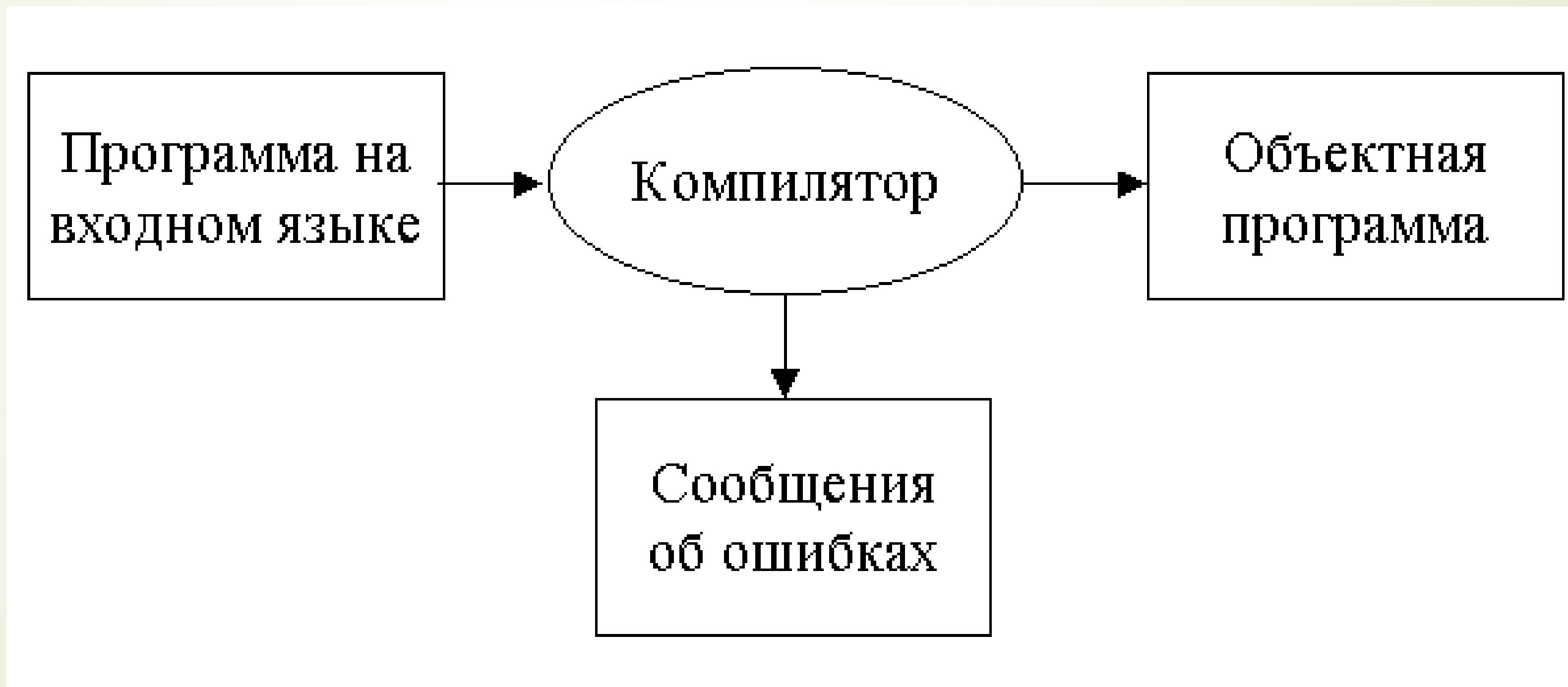
Процедурные/Объектно-ориентированные/…

Развитие языков программирования

- 1948-й ассемблер для IBM 701
- 1960-е Fortran, Algol, Basic, Cobol, Lisp
- 1970-е Pascal, C, Simula, Smalltalk, Prolog
- 1980-е C++, Object Pascal, Ada
- 1990-е Java, PERL, Delphi, Ruby
- 2000-е C#
- 2003-й Scala
- 2008-й Python
- 2010-й Kotlin

Компиляция программы

Компиляция — сборка программы, включающая трансляцию программы, написанной на исходном языке программирования высокого уровня в эквивалентную исполняемую программу на низкоуровневом языке



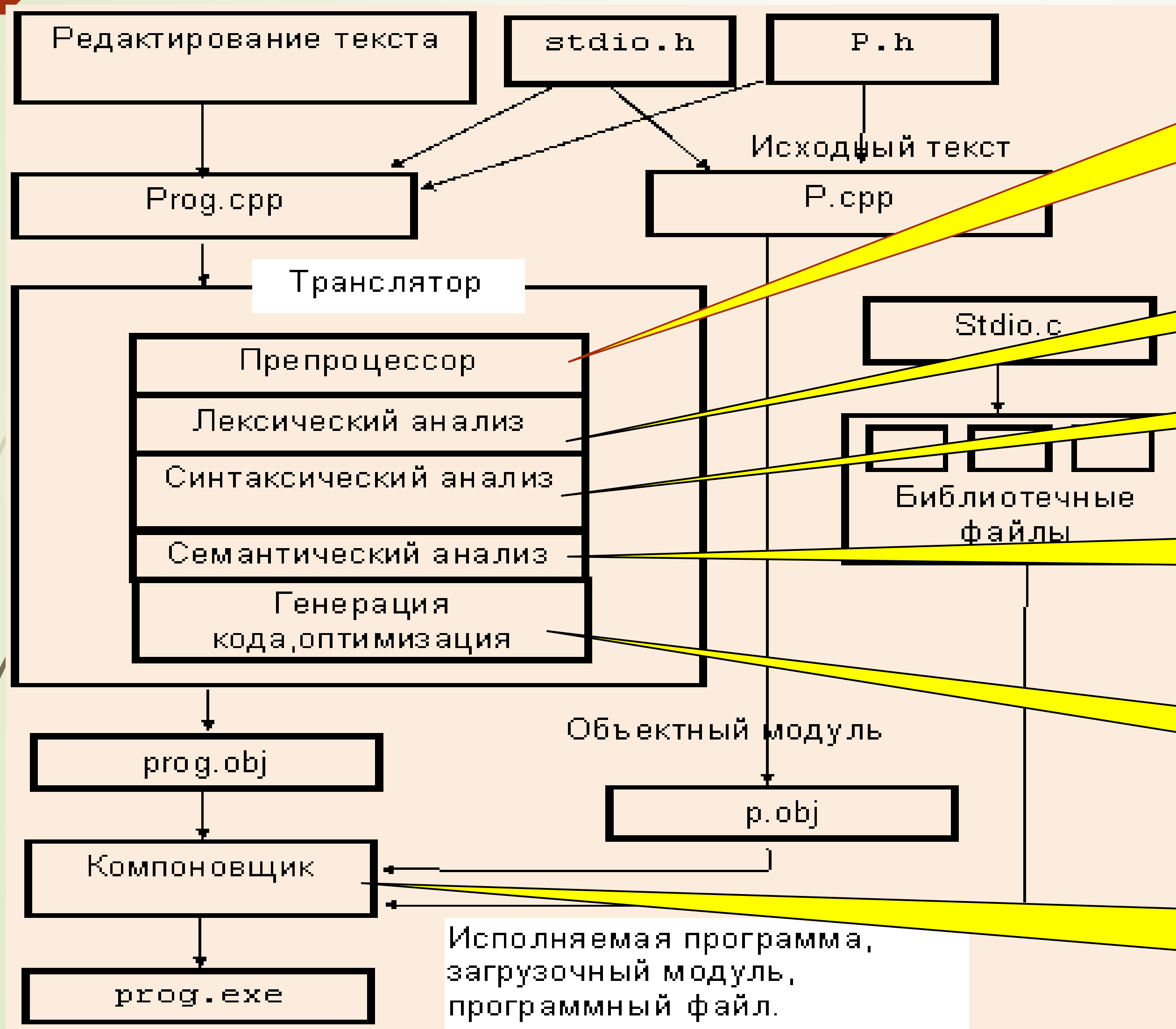
Трансляция

- **Трансляция программы** — преобразование программы, представленной на одном из языков программирования, в программу на другом языке.
- Язык, на котором представлена входная программа, называется *исходным языком*, а сама программа — *исходным кодом*. Выходной язык называется целевым языком, а выходная (результирующая) программа — *объектным кодом*.

Трансляция включает в себя:

- Лексический анализ. последовательность символов исходного файла преобразуется в последовательность лексем.
- Синтаксический анализ. Последовательность лексем преобразуется в дерево разбора.
- Семантический анализ. Дерево разбора обрабатывается с целью установления его семантики (смысла) — например, привязка идентификаторов к их декларациям, типам, проверка совместимости, определение типов выражений.
- Оптимизация. Выполняется удаление излишних конструкций и упрощение кода с сохранением его смысла.
- Генерация кода. Из промежуточного представления порождается код на целевом машинно-ориентированном языке.

Компиляция



Выполнение директив препроцессора

Проверка правописания команд

Проверка синтаксиса

Смысловая проверка конструкций (выражений)

Преобразование в объектный код

Сборка программы из объектных модулей и библиотек

Инструменты разработчика ПО

В процессе программирования в настоящее время широко используются интегрированные среды разработки (англ. Integrated development environment — IDE), в состав которых обычно входят:

- редактор кода для ввода и редактирования текста программ
- отладчик для отладки (поиска и устранения ошибок)
- транслятор для преобразования текста программы в машинное представление
- компоновщик для сборки программы из нескольких модулей
- другие служебные модули и инструменты

Интегрированная среда разработки

IDE обычно предназначены для нескольких языков программирования

- IntelliJ IDEA
- NetBeans
- Eclipse
- Qt Creator
- **Code::Blocks**
- Xcode
- Microsoft Visual Studio

Существуют специализированные IDE для одного языка программирования

- Visual Basic
- Delphi
- Dev-C++.

Этапы решения творческих задач

Решение любой задачи является творческим процессом, который состоит из нескольких последовательных этапов:

- Анализ постановки задачи и ее предметной области
- Понимание постановки и требований исходной задачи, определение предметной области, для которой поставлена задача,
- Анализ предметной области, выявление данных, которые фиксируют входную и выходную информацию (определение их структуры и свойств)
- Определение отношений между данными, условий и ограничений, накладываемых на эти отношения.

Понятие алгоритма. Свойства алгоритмов.

- Слово «алгоритм» появилось в 9-м веке и связано с именем математика Аль-Хорезми, который сформулировал правила выполнения четырех арифметических действий над многозначными числами.

- **Алгоритм** – это точно определенная последовательность действий для некоторого исполнителя, выполняемых по строго определенным правилам и приводящих через некоторое количество шагов к решению задачи.

Виды представления алгоритмов.

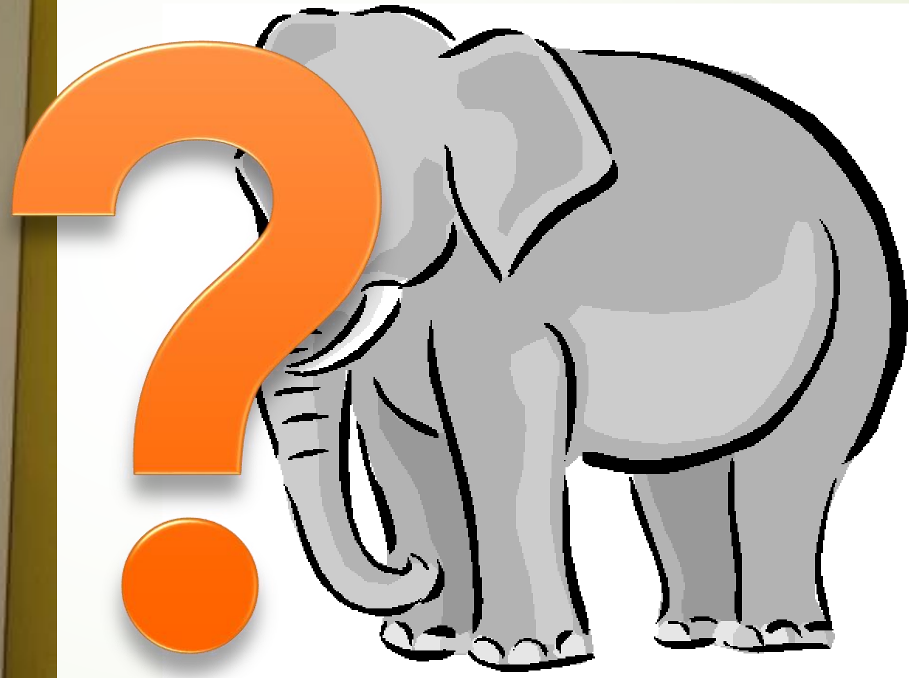
Алгоритм, реализующий решение задачи, можно представить различными способами:

- в виде текстового описания на естественном языке;
- с помощью графического описания;
- в виде таблицы значений
- на языке программирования или другом формальном языке

Свойства алгоритма:

- **Определенность** – выполнив очередное действие, исполнитель должен точно знать, что ему делать дальше.
- **Дискретность** – прежде, чем выполнить определенное действие, надо выполнить предыдущее.
- **Массовость** – по одному и тому же алгоритму решаются однотипные задачи и неоднократно.
- **Понятность** – алгоритм строится для конкретного исполнителя человеком и должен быть ему понятен. Это облегчает его проверку и модификацию при необходимости .
- **Результативность** – алгоритм всегда должен приводить к результату.

Как положить слона в холодильник?



Алгоритм решения

- Шаг 1. Берем слона
- Шаг 2. Открываем холодильник
- Шаг 3. Кладем слона в холодильник
- Шаг 4. Закрываем холодильник

Ура! Слон в холодильнике!

Задача решена!

Визуальное представление линейного алгоритма

Операции

- Отдельные действия, составляющие алгоритм, называются операциями. При этом под операцией понимается как какое-то единичное действие, например, сложение, так и группа взаимосвязанных действий.
- Операция = Операнды + Операторы
- Операнд – аргумент операции, данные, которые обрабатываются командой.
- Оператор – выполняемое действие.
- C = A + B;   (=, +: операторы, C, A, B: операнды)
- В языке C символом конца операции служит точка с запятой ;

Переменные

- Переменная это именованная область в памяти, к которой можно обращаться по имени, для выполнения различных операций над ее содержимым.
- Прежде чем использовать переменную, нужно ее определить и инициализировать (присвоить значение).
- Определение переменной: ***тип имя1, имя2, имя3;***
- Значение в переменную записывается с помощью операции присвоения = (Переменная = Выражение)

Пример:

```
char c;
short s;
int a, b;
a = 299792458;
c = 'A';
s = 258;
```

c
'A'

s
258

a
299792458

b

Имена переменных

- Имена переменных составляются из букв и цифр; первым символом должна быть буква.
- Символ подчеркивания "_" считается буквой;
- Большие (прописные) и малые (строчные) буквы различаются.
- Обычно в программах на Си малыми буквами набирают переменные, а большими — именованные константы.
- Ключевые слова if, else, int, float и т. д. зарезервированы, и их нельзя использовать в качестве имен переменных.
- Разумно давать переменным осмысленные имена в соответствии с их назначением

Типы данных языка С

Типы данных языка С

Типы данных определяют что и в каком диапазона можно записать в переменную данного типа, и какие операции можно над ней совершать.

Базовые типы данных:

- Целочисленные: short, int, long
- Вещественные: float, double
- Символьные: char

Эти типы данных являются знаковыми типами (можно записать как положительные значения, так и отрицательные).

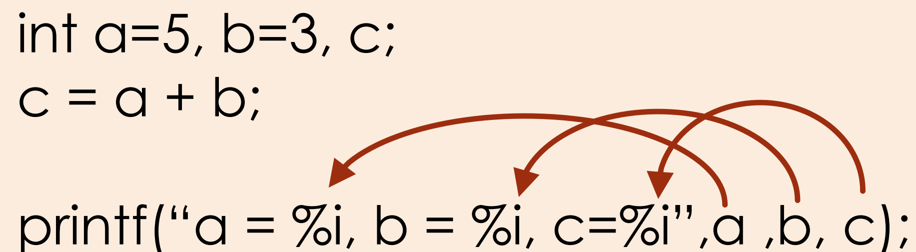
Если к имени целочисленного типа добавить ключевое слово unsigned, тогда в такую переменную можно будет записывать только положительные значения, но максимальное положительное число будет в два раза больше, чем при использовании аналогичного знакового типа.

Пример программы на языке С.

```
#include <stdio.h>
void main()
{
    printf("Hello world!");
}
```

Вывод на экран (функция printf)

- Функция printf выводит текст на экран.
- Функция printf находится внутри стандартной библиотеки ввода/вывода stdio
- В простейшем случае printf просто выводит на экран тот текст, который помещен внутрь этой функции в двойных кавычках: printf("Hello world!");
- В этот текст также можно помещать специальные управляющие последовательности, влияющие на отображение текста (\n – перейти на новую строку, \t – табуляция (вставить несколько пробелов)
- Кроме того, с помощью функции printf можно выводить на экран содержимое переменных. Для этого внутри текста помещают специальные символы (%буква), а после закрытия кавычек через запятую перечисляют переменные, чье содержимое нужно вывести на экран. Содержимое переменных будет вставлено на место этих спецсимволов.



Вывод:

a = 5, b = 3, c = 8

Вывод на экран (спецификаторы функции printf)

- Функция printf выводит на экран значения переменных различных типов, которые отформатированы согласно заданному шаблону. Этот шаблон определяется составленной по специальным правилам строкой (форматной строкой).
- *printf("текст%[флаги][ширина][.точность][размер]спецификатор", переменная)*
- С помощью спецификатора указывают переменную какого типа нужно вывести.

Спецификатор	Значение
%c	символ
%d %i	целое десятичное число
%ld	long (длинное целое)
%f	десятичное число с плавающей запятой xx.xxxx
%o	целое в восьмеричной системе счисления
%s	строка символов
%u	беззнаковое десятичное число
%x, %X	целое в шестнадцатеричной системе счисления

Вывод на экран (точность вывода функции printf)

- Для правильного вывода на экран букв кириллицы нужно предварительно вызвать функцию SetConsoleOutputCP(1251) из библиотеки windows или функцию setlocale(0, "rus") из locale.
- В функции printf можно задать точность вывода на экран вещественных значений (число знаков после запятой). Для этого между % и спецификатором типа нужно написать .число
- *printf("%[.точность]спецификатор", переменная);*

Пример:

```
float f1, f2, f3;
f1 = 10;
f2 = 3;
f3 = f1/f2;
printf("result = %.2f", f3); //выведет содержимое переменной f3 с точностью до двух знаков после запятой.
```

Вывод на экран (ширина вывода функции printf)

- В функции printf можно задать минимальную фиксированную ширину поля для вывода значений. Для этого между % и спецификатором типа нужно написать целое число: *printf("%[ширина]спецификатор", переменная);*
- Если значение которое нужно вывести больше, чем ширина поля, то вывод выйдет за пределы поля, если оно меньше, то вывод будет дополнен пробелами слева.
- В модификаторах ширины и точности вместо конкретных значений можно указать символ * (звездочка), в таком случае конкретное значение нужно будет указать в списке параметров перед значением для вывода.

Пример 1:

```
int i, j, k;
i = 10;
j = 100;
k = 1000;
printf("%5i%5i%5i\n", i, j, k);
printf("%5i%5i%5i", k, j, i);
```

Пример 2:

```
int b;
float f, a;
a = 5.4;
b = 3;
f = a/b;
printf("%.*f", b, f);
```

Размер и диапазон целочисленных типов данных

- С помощью команды sizeof(тип данных) можно узнать сколько байт в памяти занимает переменная того или иного типа данных.
- Команда sizeof(char) вернет в качестве ответа 1, потому что char занимает в памяти 1 байт.
- Байт это 8 бит, ячеек в каждую из которых может быть записан только 0 или 1
- Биты нумеруются справа налево, начиная с нуля (от индекса 0 справа, до 7 слева)
- В знаковых типах самый левый бит используется для определения знака числа (+-)
- Если в знаковом бите записан 0 то это положительное число, если 1 то отрицательное.
- Отрицательное число записывается в дополнительном коде (инвертируются все биты и прибавляется единица)
- Максимальное число получается когда во всех битах записана 1

7	6	5	4	3	2	1	0
знак	1	1	1	1	1	1	1

- $1111111_2 + 1_2 = 10000000_2 = 2^7 = 128 \Rightarrow 1111111_2 = 127$
- char принимает значения от -128 до 127
- unsigned char (char без знака) принимает значения от 0 до 255

Базовые операции. Приоритет операций

<table>
<tr><th>Операторы</th><th>Значение</th><th>Выполняются</th></tr>
<tr><td>() [] -> .</td><td>Обращение к элементу</td><td>слева направо</td></tr>
<tr><td>! ~ ++ -- + - * & (тип) sizeof</td><td>Инкремент/знак числа</td><td>справа налево</td></tr>
<tr><td>* / %</td><td>Умножение/деление/остаток</td><td>слева направо</td></tr>
<tr><td>+ -</td><td>Сложение/вычитание</td><td>слева направо</td></tr>
<tr><td><< >></td><td>Битовый сдвиг</td><td>слева направо</td></tr>
<tr><td>< <= > >=</td><td>Проверка неравенства</td><td>слева направо</td></tr>
<tr><td>== !=</td><td>Проверка равенства</td><td>слева направо</td></tr>
<tr><td>&</td><td>Побитовое И</td><td>слева направо</td></tr>
<tr><td>^</td><td>Побитовое исключающее ИЛИ</td><td>слева направо</td></tr>
<tr><td>|</td><td>Побитовое ИЛИ</td><td>слева направо</td></tr>
<tr><td>&&</td><td>Логическое И</td><td>слева направо</td></tr>
<tr><td>||</td><td>Логическое ИЛИ</td><td>слева направо</td></tr>
<tr><td>?:</td><td>Тернарный оператор</td><td></td></tr>
<tr><td>= += -= *= /= %= &= ^= |= <<= >>=</td><td>Присвоение</td><td>справа налево</td></tr>
<tr><td>,</td><td>Связка нескольких выражений</td><td>слева направо</td></tr>
</table>

Сокращенная запись выражений присвоения

- Некоторые операции присвоения можно записать в сокращенном виде. Такая форма записи широко распространена при написании программ на С.

Сокращение	Полная запись	Название
i++	i = i + 1	Инкремент
i--	i = i - 1	Декремент
i+=n	i = i + n	
i-=n	i = i – n	
i*=n	i = i * n	
i/=n	i = i/n	
i%=n	I = i%n	

Битовый сдвиг

- Операторы << и >> сдвигают соответственно влево или вправо биты двоичного представления своего левого операнда на число битовых позиций, задаваемое правым операндом, который должен быть неотрицательным.
- При сдвиге влево освобождающиеся биты заполняются нулями.
- При сдвиге вправо беззнаковой величины освобождающиеся биты заполняются нулями.
- При сдвиге вправо знаковой величины возможны два варианта:
  - Распространение знака ("арифметический сдвиг")
  - Заполнение освобождающихся разрядов нулями ("логический сдвиг").

```
//Пример сдвига влево:
int x, y;

x = 3; // двоичное представление 0011
y = x << 1; // 0110  или 6
y = x << 2; // 1100 или 12
```

```
//Пример сдвига вправо:
int x, y;

x = 10; // двоичное представление 1010
y = x >> 1; // 0101  или 5
y = x >> 2; // 0010 или 2
```

Побитовое ИЛИ и побитовое И

- Побитовые операции | (или) и & (и) работают с двоичным представлением двух целых чисел. Результатом является третье целое число, каждый бит которого получен выполнением операции над соответствующими битами двух операндов.

c = a & b		
a	**b**	**c**
0	0	0
0	1	0
1	0	0
1	1	1

```
//Пример побитового И:
int x, y, z;

x = 3; // двоичное представление 0011
y = 6; // двоичное представление 0110
z = x & y; //                  z = 2  <=     0010
```

c = a | b		
a	**b**	**c**
0	0	0
0	1	1
1	0	1
1	1	1

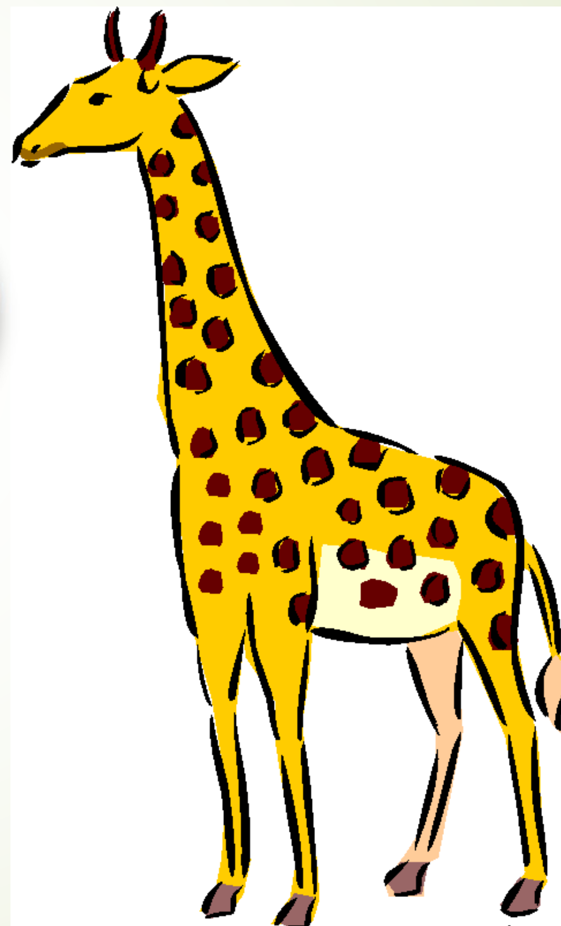
```
//Пример побитового ИЛИ:
int x, y, z;

x = 3; // двоичное представление 0011
y = 6; // двоичное представление 0110
z = x | y; //                  z = 7  <=     0111
```

Представление в памяти вещественных чисел

- Для хранения информации о вещественном числе используется экспоненциальная запись, состоящая из двух частей: мантиссы и порядка (2.5E3 = 2500)
- Число записывается в нормализованной форме: перед дробной частью находится только одна цифра.
- **Мантисса** - дробная часть числа, **порядок** - на сколько сдвинута дробная часть.
- Перевод вещественного числа в двоичное представление (стандарт IEEE-745):
  - Целая часть вещественного числа переводится в двоичную систему счисления
  - Дробная часть вещественного числа умножается на 2, берется целая часть, это первое значения после запятой, остаток опять умножается на 2, и т.д.
  - В получившемся результате дробная часть сдвигается, так чтобы перед ней была только одна единица.
  - Записывается новая дробная часть и порядок в виде: сдвиг + 127 (нужно чтобы сдвиг мог быть отрицательным)

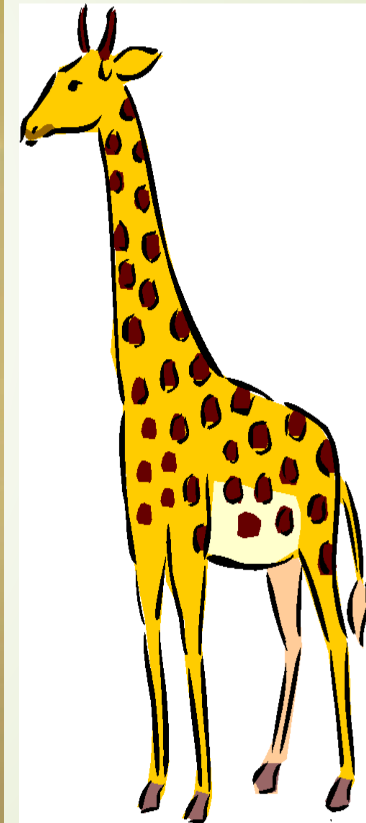
	зн	порядок (степень)								мантисса (дробная часть)																						
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
inf	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
nn	0	1	1	1	1	1	1	1	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
2.5	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Как положить жирафа в холодильник?

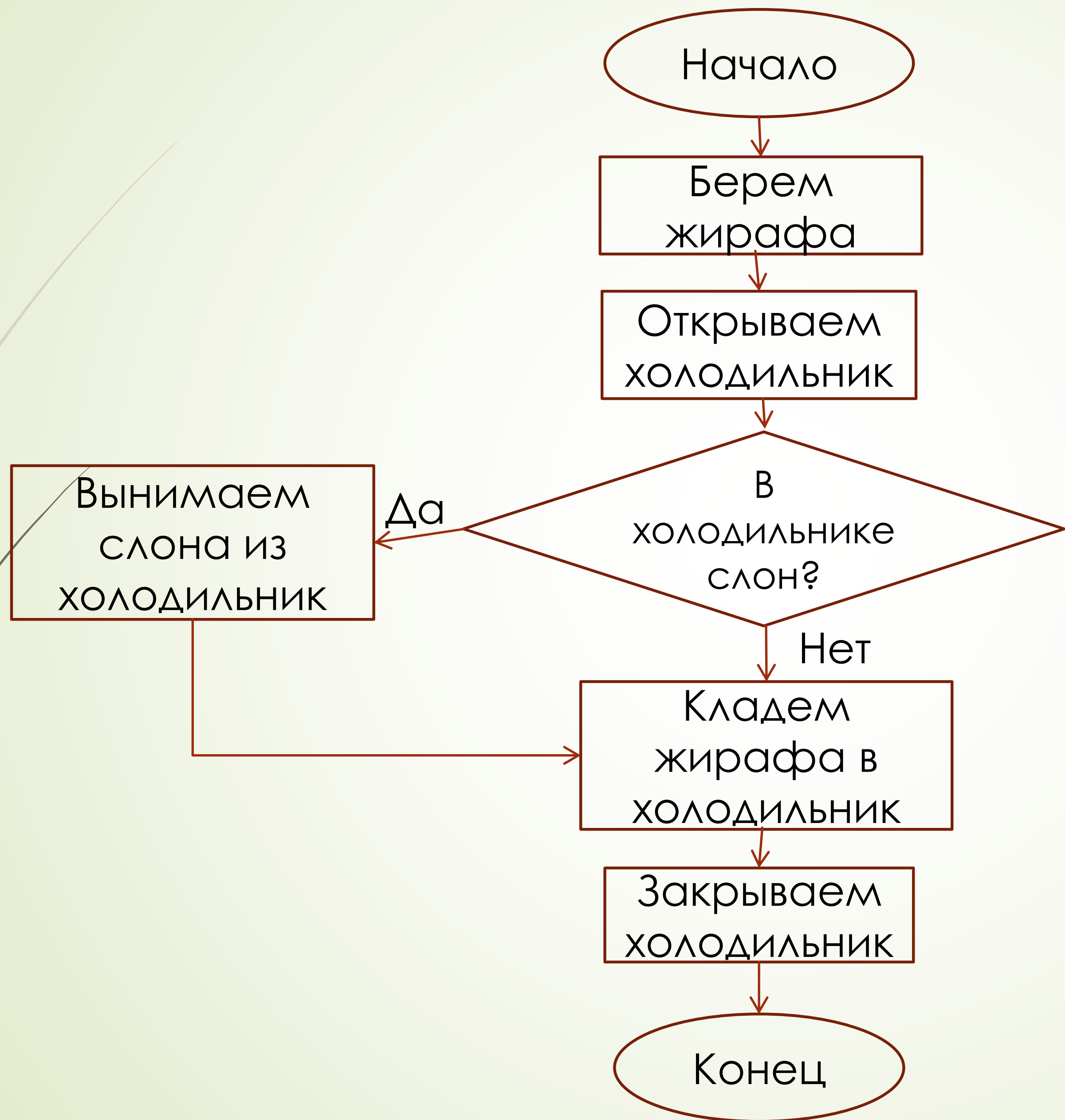


Алгоритм решения

- Шаг 1. Берем жирафа
- Шаг 2. Открываем холодильник
- Шаг 3. Вынимаем слона из холодильника
- Шаг 4. Кладем жирафа в холодильник
- Шаг 5. Закрываем холодильник



Визуальное представление алгоритма с ветвлением



Оператор ветвления if

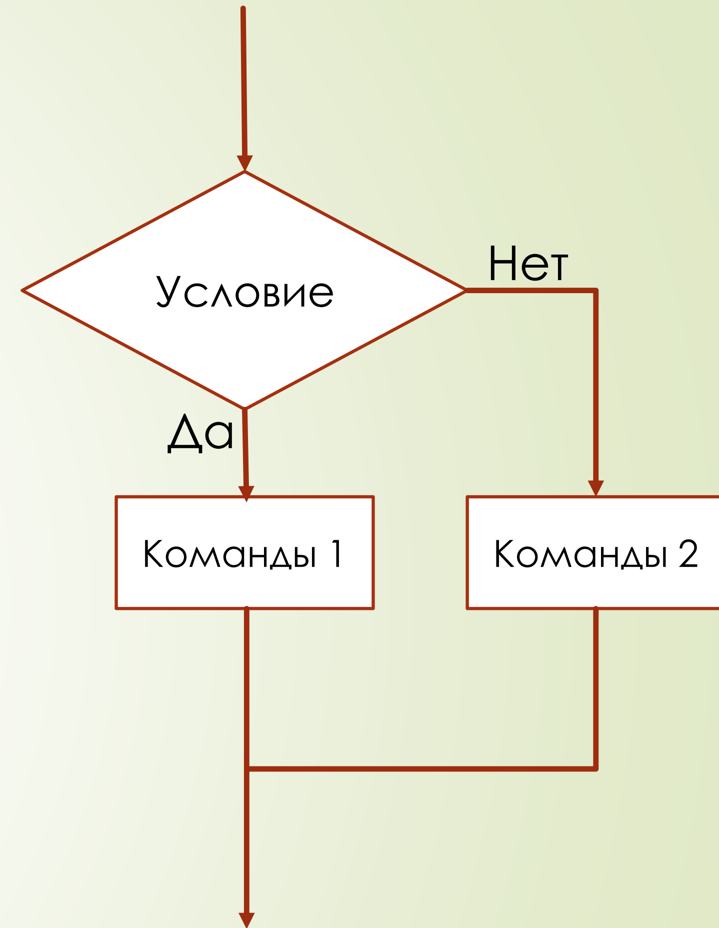
Форма операции ветвления:

if (Логическое условие)

Блок команд если условие выполнилось

[else

Блок команд если условие не выполнилось]



Пример:

```
int i=7;
if (i>0)
    printf("i is positive");
else
    printf("i is not positive");
```

Генерация псевдослучайных чисел

- Функция rand() возвращает случайное целое положительное число в диапазоне от 0 до RAND_MAX.
- Чтобы возвращать случайное значение в диапазоне от min до max включительно следует использовать остаток от деления по формуле

  rand()%(max-min+1)+min;
- Также следует инициализировать генератор случайным смещением, чтобы при каждом запуске выдавались другие значения

  srand (time(NULL));

Операторы цикла

- Если какой-то набор операций необходимо выполнить несколько раз, то эти операции помещают внутрь оператора цикла.
- Операции находящиеся внутри оператора цикла повторяются до тех пор, пока условие цикла не перестанет быть истинным.
- Условие цикла это какое-то логическое выражение (например сравнение двух переменных) результат которого это истина или ложь.

Операторы цикла

Оператор for

for (инициализация; логическое условие; инкремент)

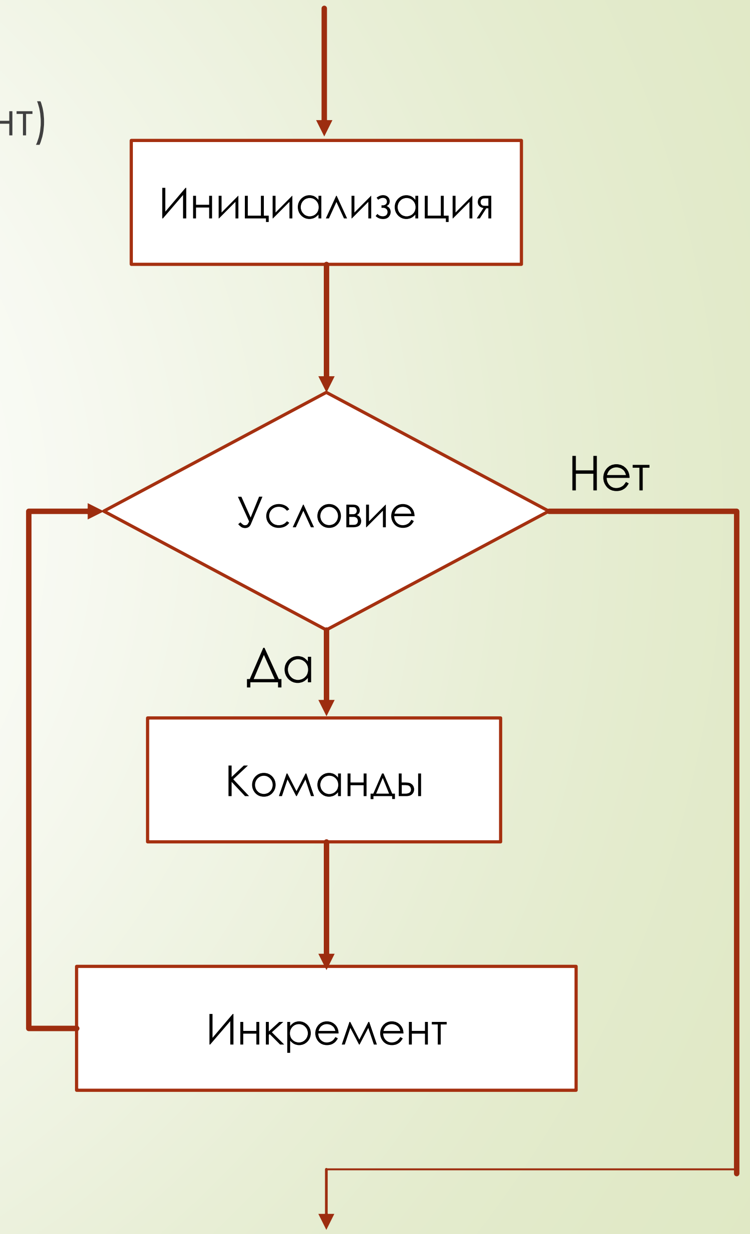
Блок повторяющихся команд

Пример:

```
int i;
for (i=0; i<4; i++)
   printf("Hello world\n");
```

Оператор while

```
инициализация;
while (логическое условие)
{
  …
  инкремент;
}
```



Область видимости переменных

- Блок это последовательность из нескольких операций заключенная в фигурные скобки { }
- Операции заключенные в один блок воспринимаются как единое целое операторами ветвления и цикла.
- Переменная видна и может быть использована только в том блоке, в котором она определена, и в блоках уровнями ниже.

Пример:

```
int i, j, sum;
for (i=1; i<10; i++)
{
    int mul;
    for (j=1; j<10; j++)
    {
        sum = i+j;
        mul = i*j;
        printf("%i + %i = %i, %i x %i = %i\n", i, j, sum, i, j, mul);
    }
}
printf("sum = %i", sum);
printf("mul = %i", mul);        //ошибка. эта переменная здесь не видна
```

Стиль оформления кода

- Для объявления идентификаторов состоящих из нескольких слов используется:
  - Camel-регистр: CreateWindow, createWindow
  - Snake-регистр: MAX_ARRAY_SIZE, fill_array
- Идентификаторы могут иметь префиксы, поясняющие их тип или смысл (венгерская нотация): *sClientName, lAmount, ix*
- Идентификатор должен быть понятным и читаемым вслух. Для стандартных задач допускаются однобуквенные идентификаторы: *i, j – счетчики циклов.*
  - Хорошо: i; count; buffer_length; argSum;
  - Плохо: asdghj; lnoftmpbuf; wrtlnscr;
- Новый оператор – новая строка, вложенные операторы – правее на один отступ.
- Одинаковый уровень вложенности должен быть равноудалён от левого края экрана.
- Открывающая скобка блока обычно пишется под заголовком блока, тогда открывающая и закрывающая симметричны, но открывающая фигурная скобка иногда помещается в конце строки с заголовком блока.

Занятие 2. Темы

- Преобразование типов
- Математические функции
- Ввод данных
- Работа с циклами

Преобразования типов

- Если операнды оператора принадлежат к разным типам, то они приводятся к некоторому общему типу. Результат операции также будет этого типа.
- Обычно автоматически производятся лишь те преобразования, которые без какой-либо потери информации превращают операнды с меньшим диапазоном значений в операнды с большим диапазоном, как, например, преобразование целого в число с плавающей точкой в выражении вроде f+i.
- Выражения, не имеющие смысла, например число с плавающей точкой в роли индекса, не допускаются.
- Выражения, в которых могла бы теряться информация (скажем, при присваивании длинных целых переменным более коротких типов или при присваивании значений с плавающей точкой целым переменным), могут повлечь за собой предупреждение, но они допустимы.
- Значения типа char — это просто малые целые, и их можно свободно использовать в арифметических выражениях, что значительно облегчает всевозможные манипуляции с символами.

Правила преобразования типов

А операция В

- Если какой-либо из операндов принадлежит типу long double, то и другой приводится к long double.
- В противном случае, если какой-либо из операндов принадлежит типу double, то и другой приводится к double.
- В противном случае, если какой-либо из операндов принадлежит типу float, то и другой приводится к float.
- В противном случае операнды типов char и short приводятся к int.
- Если один из операндов типа long, то и другой приводится к long.

Оператор приведения типа

- Имя типа, записанное перед унарным выражением в скобках, вызывает приведение значения этого выражения к указанному типу.

  (имя-типа) унарное выражение

- Данная конструкция называется приведением

Пример:

```
int x=5, y=2;
float result;
result = x/y;
result = (float)x/y;
printf("result = %.2f", result);
```

Математические функции

В подключаемой библиотеке <math.h> описываются математические функции.

x и y имеют тип double, n — тип int, и все функции возвращают значения типа double. Углы в тригонометрических функциях задаются в радианах.

Функция	Значение
sin(x)	синус x
cos(x)	косинус x
tan(x)	тангенс x
exp(x)	экспоненциальная функция e^x
log(x)	натуральный логарифм ln(x), x > 0
log10(x)	десятичный логарифм log_{10}(x), x > 0
pow(x, y)	возведение в степень x^y
sqrt(x)	корень квадратный из x, x > 0
ceil(x)	наименьшее целое в виде double, которое не меньше x
floor(x)	наибольшее целое в виде double, которое не больше x
fabs(x)	абсолютное значение |x|

Ввод данных (функция scanf)

Функция scanf, обеспечивает ввод данных с клавиатуры и по форме похожа на printf.

Функция scanf читает символы до разделителя (пробел или перевод строки) из стандартного входного потока, интерпретирует их согласно спецификациям строки и записывает результаты в свои остальные аргументы.

scanf("%формат", &переменная);

Формат чтения:

- %c чтение символа
- %d чтение десятичного целого
- %i чтение десятичного целого
- %f чтение вещественного числа (с плавающей запятой, float)
- %lf чтение вещественного числа большого диапазона (double)
- %h чтение короткого целого (short)
- %o чтение восьмеричного числа
- %s чтение строки
- %x чтение шестнадцатеричного числа

Для правильного ввода с клавиатуры букв кириллицы нужно предварительно вызвать функцию SetConsoleCP(1251) из библиотеки windows.

Занятие 3. Темы

- Оператор ветвления switch
- Массивы
- Объявление массива
- Инициализация массива
- Прерывание цикла. Операторы break и continue
- Двумерный массив
- Трехмерный массив
- Массив символов (строка)

Оператор ветвления switch

- Инструкция switch используется для выбора одного из многих путей. Она проверяет, совпадает ли значение выражения с одним из значений, входящих в некоторое множество целых констант, и выполняет соответствующую этому значению ветвь программы.
- В конце каждого пункта case ставят оператор break, иначе после нахождения нужного case программа начнет выполнять и все последующие пункты case тоже.

Вид оператора switch:

```
switch (выражение)
{
case конст-выр: инструкции
case конст-выр: инструкции
default: инструкции
}
```

Оператор ветвления switch

Пример оператора switch:

```
int n;
scanf("%i", &n);
switch (n)
{
  case 1:                //если в переменной n записана 1, то сделай это:
    printf("one");
    break;
  case 2:                //если в переменной n записана 2, то сделай это:
    printf("two");
    break;
  case 3:                //если в переменной n записана 3, то сделай это:
    printf("three");
    break;
  default:               //если в n записана не 1, не 2, и не 3 то сделай это:
    printf("unknown number");
}
```

Массивы

- Массив это упорядоченный набор данных, используемый для хранения данных одного типа.
- Массив имеет постоянную длину и хранит единицы данных одного и того же типа.
- Массив относится к структурам данных с произвольным доступом.
- Доступ к отдельному элементу массива осуществляется по его индексу.
- Работа со всеми элементами массива осуществляется с помощью циклов.
- В языке С элементы массива всегда нумеруются начиная с нуля.
- Размерность массива — это количество индексов, необходимое для однозначной адресации элемента в рамках массива. По количеству используемых индексов массивы делятся на одномерные, двумерные, трёхмерные и т. д.

Объявление массива

- Объявление одномерного массива: ***тип имя[размер];***

Пример:

Объявление массива: ***int array[10]***

Создается массив из десяти ячеек, в каждую из которых можно записать целое число (int).

- *Запись отдельного элемента массива:* ***имя_массива[индекс];***
- В языке С элементы массива всегда нумеруются начиная с нуля.

Пример:

array[8] = 5;

array	0	1	2	3	4	5	6	7	8	9
	int	int	int	int	int	int	int	int	5	int

Записывает число 5 в ячейку номер 8 (девятую ячейку от начала)

Внимание! Язык С не проверяет индекс массива на допустимость. В случае неправильного индекса программа вернет непредсказуемое значение из памяти.

Проверка индекса на корректность ложится на плечи программиста.

Инициализация массива

- Массив можно инициализировать значениями при объявлении. Для этого нужно поставить знак присвоения и в фигурных скобках перечислить значения, которые должны быть записаны в массив.
- Если указать меньше элементов чем длина массива, то оставшиеся элементы будут заполнены неопределенными значениями.
- Если не указать размер массива, то он будет высчитан по числу значений в фигурных скобках.
- В дальнейшем можно изменить содержимое массива, но изменить размер массива после его объявления уже нельзя.

Пример:

```
int arr[10]={5,4,17,1,8};   //создали массив из 10 элементов, но заполнили только 5.
int arr2[] = {6,14,9,2,7};   //создали массив из 5 элементов
```

Прерывание цикла. Оператор break

- Для досрочного выхода из цикла можно использовать уже знакомый оператор **break**.
- **break** прерывает выполнение цикла и программа продолжается выполнением тех операций которые идут после цикла.
- Если в программе один цикл вложен в другой то **break** приводит к выходу только из того цикла в котором он непосредственно находится.

Пример:

```
int i, x=5;
int arr[] = {3,8,5,6,1,5,9,4};

for (i=0; i<8; i++)
  if (arr[i]==x)
      break;
printf("first %i is with index %i", x, i);
```

x	i	i<8	arr[i]	arr[i]==x
5	0	да	3	нет
5	1	да	8	нет
5	2	да	5	да

first 5 is with index 2

Прерывание цикла. Оператор continue

- Оператор **continue** вызывает досрочное завершение текущей итерации (прохода) цикла и переход к следующей.
- Те операции которые находятся между оператором **continue** и концом цикла на текущей итерации пропускаются.

Пример:

```
int i;
int arr[] = {3,8,5,6,1,5};

printf("only odd numbers:\n");
for (i=0; i<6; i++)
{
  if (arr[i]%2==0)
      continue;
  printf("%i ", arr[i]);
}
```

i	i<6	arr[i]	arr[i]%2==0	printf
0	да	3	нет	3
1	да	8	да	
2	да	5	нет	5
3	да	6	да	
4	да	1	нет	1
5	да	5	нет	5
6	нет			

Двумерный массив

- Двумерный массив можно представить как таблицу состоящую из строк и столбцов.
- В объявлении двумерного массива первый (левый) индекс принято читать как количество строк, а второй (правый) как количество столбцов.
- Объявление двумерного массива: ***тип имя[строки][столбцы];***

Двумерный массив matrix				
			8	

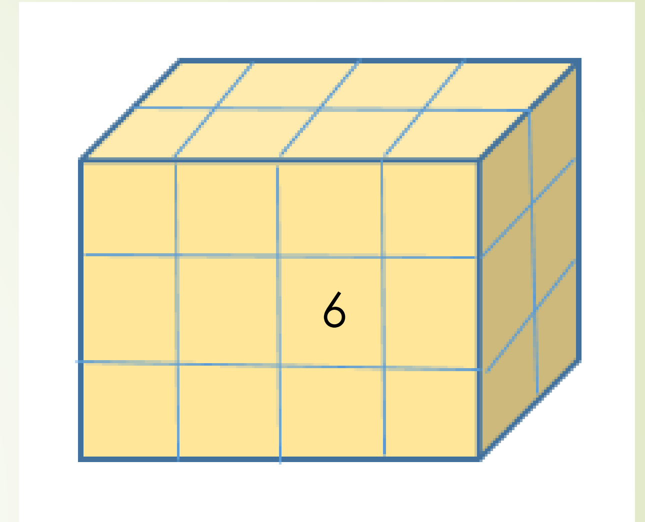
Пример:
Объявление массива: ***int matrix[4][5];***
Создается массив из четырех строк и пяти столбцов. В каждую ячейку этого массива можно записать целое число (int).

Запись значения в ячейку: ***matrix[2][3]=8;***
Двумерный массив также можно инициализировать значениями при объявлении. При этом каждую строку нужно заключить в отдельные фигурные скобки.
Пример:
int matrix[4][5]={{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15},{16,17,18,19,20}};
int matrix[][]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};

Трехмерный массив

- Трехмерный массив можно представить как кубик состоящий из высоты, ширины и глубины.
- Объявление трехмерного массива:

  тип имя[глубина][строки][столбцы];



Пример:
Объявление массива: ***int cube[2][3][4];***
Создается массив глубиной два из трех строк и четырех столбцов.
В каждую ячейку этого массива можно записать целое число (int).

Запись значения в ячейку: ***cube[0][1][2]=6;***
Трехмерный массив также можно инициализировать значениями при объявлении. При этом каждую строку и каждый двумерный массив нужно заключить в отдельные фигурные скобки.

Пример:
```
int cube[2][3][4] = {{{1,2,3,4}, {5,6,7,8}, {9,10,11,12}},
                     {{13,14,15,16}, {17,18,19,20}, {21,22,23, 24}}};
```

Массивы и строки

- В языке С нет отдельного типа данных для хранения текста (строк). Для их хранения используется **массив символов** (char).
- Символом окончания строки является записанный в ячейку спецсимвол‘\0’.
- В такой массив нельзя записать значение с помощью операции присвоения. Чтобы поместить текст в массив нужно использовать функцию strcpy.
- Функция strcpy копирует один массив в другой: *strcpy(куда, откуда)*.
- Ввод данных в такой массив с клавиатуры и вывод на экран осуществляется с помощью спецификатора %s
- Функция scanf() считывает строку только до пробела. Если нужно считать с клавиатуры строку включающую в себя пробелы то нужно использовать другие функции, например gets(): *gets(массив_в_который_записать)*

Пример

```
char name[10];
printf("What is your name?\n");
scanf("%s", &name);
printf("Hello, %s! Nice to meet you!\n“, name);
```

```
char city[10];
strcpy(city, "Nice");
printf("%s is a nice city",city);
```

Занятие 4. Темы

- Работа с файлами. Принцип сэндвича
- Открытие файла. Путь и режим доступа
- Запись в текстовый файл. Функция fprintf
- Чтение из текстового файла. Функция fscanf
- Запись в двоичный файл. Функция fwrite
- Чтение из двоичного файла. Функция fread
- Перемещение по файлу. Функция fseek

Работа с файлами

- В языке С работа с файлами состоит как сэндвич из трех частей.
  1. Открытие файла.
  2. Работа с файлом.
  3. Закрытие файла.

- Для связи программы с файлом нужно создать специальную переменную, ее тип указатель на файл: **FILE***

Пример

FILE fp;*

Открытие и закрытие файла

- Открытие файла на чтение либо на запись осуществляется с помощью функции **fopen**.
- Вид функции fopen: *fopen(имя_файла_включая_путь, режим_доступа к файлу)*
- Результат выполнения функции **fopen** присваивается указателю на файл для дальнейшей работы с открытым файлом.

Пример:

FILE fp;*

fp = fopen("file.txt" , "w");

- Закрытие файла осуществляется с помощью вызова функции fclose, которой в качестве параметра передается указатель на открытый файл.

Пример: fclose(fp);

Путь к файлу

- Первый параметр функции fopen должен содержать имя файла и путь к нему.
- Путь к файлу может быть двух видов:
  1. Относительный
  2. Абсолютный
- Относительный путь определяет местоположение файла относительно расположения исходного кода программы.
  1. Если указано только имя файла, то он создается в той же папке что и файл с исходным кодом программы.
  2. Символы ../ говорят о том, что нужно подняться на одну папку выше в иерархии.

Пример: fp = fopen("../moveup.txt" , "w");

- Абсолютный путь указывает полный путь к расположению файла, включая диск и папки. Путь должен существовать.

Пример: fp = fopen("D:/My Documents/fullpath.txt" , "w");

Режим доступа к файлу

Режим доступа определяет какие операции можно совершать с открытым файлом.

Режим	Значение
r	Открытие файла для чтения. Файл должен существовать.
w	Создание пустого файла для записи. Если файл с таким именем уже существует его содержимое стирается.
a	Добавление данных в конец файла. Если файл не существует то он создается.
r+	Открытия файла для чтения и записи. Файл должен существовать.
w+	Создание пустого файла для записи. Если файл с таким именем уже существует его содержимое стирается.
a+	Открытие файла для чтения и добавления данных. Данные дописываются в конец файла.

Запись в файл. Функция fprintf()

- Функция fprintf аналогична функции вывода на экран printf. Но она выводит форматированный текст не на экран, а в файл.
- У функции fprintf три параметра, первый это указатель на файл куда будет происходить запись. Этому указателю должно быть ранее присвоено значение с помощью вызова функции fopen с режимом доступа на запись.
- Два других параметра аналогичны параметрам функции printf. Форматная строка и переменные, содержимое которых будет записано.

Пример:

```
    int i;
    FILE* fp;
    fp = fopen("file.txt" , "w");

for (i=1; i<10; i++)
        fprintf(fp, "%i  ",  i);
    fclose(fp);
```

Чтение из файла. Функция fscanf()

- Функция fscanf аналогична функции чтения данных scanf. Но она читает данные не с клавиатуры, а из файла.
- У функции fscanf три параметра, первый это указатель на файл откуда будет происходить чтение. Этому указателю должно быть ранее присвоено значение с помощью вызова функции fopen с режимом доступа на чтение.
- Два других параметра аналогичны параметрам функции scanf. Формат читаемых данных и переменная, в которую будет записан результат чтения.
- После вызова функции fscanf происходит продвижения курсора в файле к следующим данным.
- Если достигнут конец файла то происходит повторное чтение последнего значения.
- Узнать что достигнут конец файла можно с помощью функции feof которой в качестве параметра передается указатель на открытый файл.

Пример чтения из файла с fscanf()

```
int x;
FILE* fp;
fp = fopen("file.txt" , "r");
while(!feof(fp))
   {
       fscanf(fp, "%i ",&x);
       printf("x = %i\n", x);
   }
fclose(fp);
```

Работа с двоичными файлами. Запись

- В двоичных файлах данные записываются как последовательность бит.
- Для открытия файла в двоичном режиме также используется функция fopen, но к режиму доступа добавляется буква b (rb, wb, ab, rb+,wb+, ab+
- Запись в двоичный файл осуществляется с помощью функции fwrite
- В двоичный файл данные записываются с помощью массива

Описание функции:

fwrite(массив, размер одного элемента в байтах , количество элементов , указатель на файл);

Пример:

```
int arr[] = {3,8,5,6,1,5,9,4};
FILE* fp;
fp = fopen("file.txt" , "wb");
fwrite(arr, 4, 8 , fp);
fclose(fp);
```

Работа с двоичными файлами. Чтение

- Чтение из двоичного файла осуществляется с помощью функции fread
- Из двоичного файла данные записываются в массив

Описание функции:

fread(массив, размер одного элемента в байтах , количество элементов , указатель на файл);

Пример:

```
int arr[10];
FILE* fp;
fp = fopen("file.txt" , “rb");
fread(arr, sizeof(int), 10, fp);
fclose(fp);
```

Перемещение по файлу. Функция fseek()

- С помощью функции fseek можно перемещаться по открытому файлу, чтобы записывать/считывать данные не с начала файла, а с произвольного места.
- У функции fseek три параметра, первый это указатель на открытый файл по которому перемещаемся, второй – на сколько байт перемещаемся, третий – откуда (от начала файла, от конца, от текущего положения): *fseek(файл, сколько, откуда)*
- Для третьего параметра функции fseek существуют буквенные обозначения:
  - SEEK_SET – начало файла
  - SEEK_CUR – текущее положение в файле
  - SEEK_END – конец файла
- Перемещаться можно как по текстовому, так и по двоичному файлу, но в текстовом файле нельзя сказать на какой элемент попадешь.

Занятие 5. Темы

- Директивы препроцессора: include, define, условия
- Функции. Объявление и определение функции
- Параметры функции и возвращаемое значения
- Передача фактических параметров в функцию
- Функции с переменным числом параметров
- Параметры функции main()
- Рекурсивные функции
- Многофайловая программа. Заголовочные файлы

Препроцессор. Директива #include

- При компиляции программы на С сначала запускается препроцессор. Он проводит первоначальную обработку программы, выполняя специальные команды (директивы) препроцессора, если они содержатся в программе.
- Наиболее часто используемые директивы препроцессора это **#include** и **#define**
- Конструкция вида ***#include "имя-файла"*** или ***#include <имя-файла>*** в результате работы препроцессора заменяется содержимым файла с соответствующим именем.
- Включаемый файл сам может содержать в себе строки #include.
- Если имя файла заключено в двойные кавычки, то файл ищется среди исходных файлов программы. Если имя файла заключено в угловые скобки < и >, то поиск осуществляется по файлам готовых библиотек.
- Часто код программы на С начинается с нескольких строк #include, ссылающихся на прототипы нужных библиотечных функций из заголовочных файлов, например <stdio.h>
- Директива #include это хороший способ собрать вместе объявления большой программы. Она гарантирует, что все исходные файлы будут пользоваться одними и теми же определениями и объявлениями переменных, благодаря чему предотвращаются особенно неприятные ошибки.

Директива #define

- С помощью директивы **#define** осуществляется макроподстановка.
- Макроподстановка используется для простейшей замены одного текста другим.
- ***#define имя замещающий-текст*** означает что во всех местах программы, где встречается лексема ***имя***, вместо нее будет помещен ***замещающий-текст*.**
- Имена в #define обычно задаются заглавными буквами, по тем же правилам, что и имена обычных переменных.
- Область видимости имени, определенного в #define, простирается от данного определения до конца файла.
- В определении макроподстановки могут фигурировать более ранние #define-определения.

Пример:

```
#define N 10
int main()
{
   int i;
   for (i=0; i<N; i++)                    //при компиляции вместо N будет подставлено число 10
   printf("Hello! %i, %i\n", i, N);
}
```

Директива #define с параметрами

- В директиве #define можно указать в круглых скобках параметры, тогда будет произведена двойная замена: имя на замещающий текст, а в замещающем тексте параметр на фактическое значение.
- Такая конструкция похожа на вызов функции, но между ними есть важное различие. При вызове функции сначала вычисляется значение параметра, и это значение передается в функцию. При использовании директивы #define происходит подстановка значений, без их вычисления, и только потом вычисляется итоговый результат.

```
#define POW2(x) (x * x)

void main()
{
  float a, b;
  a = 5;
  b = POW2(a);
  printf("%.0f^2 = %.0f\n", a, b);
  b = pow(a, 2);
  printf("%.0f^2 = %.0f", a, b);
}
```

```
#define POW2(x) (x * x)

void main()
{
  float a, b;
  a = 5;
  b = POW2(a+1);
  printf("%.0f^2 = %.0f\n", a, b);
  b = pow(a+1, 2);
  printf("%.0f^2 = %.0f", a, b);
}
```

Директива #define и условная компиляция

- С помощью директив препроцессора можно организовать так называемую условную компиляцию, при которой часть кода в зависимости от некоторого условия, исключается из компиляции и в итоговой программе не используется.
- Условная компиляция используется для создания разных версий одной программы (например подстройку под определенную операционную систему), для включения вывода отладочных сообщений, и для исключения повторного включения библиотек директивой #include.
- Значение, которое будет проверяться условной компиляцией, задается директивой #define. Есть два варианта проверки:
  - Если в **#define** задано конкретное значение, то оно проверяется с помощью директив **#if**, **#else**, **#elif**, как обычная операция ветвления. В конце блока ставится директива **#endif**, как указание на то, что здесь заканчивается код, который будет вставлен при выполнении условия.
  - Если в **#define** задано только имя без замещаемого значения, тогда проверяется само наличие такого объявления (задано ли такое имя?). Это делается с помощью директив **#ifdef** – условие выполнено если имя есть, и **#ifndef** - если такого имени нет.

Пример условной компиляции

```
#define SIZE 15

void main()
{
   int array[SIZE];
   #if SIZE < 10
   printf("Small array");
   #elif SIZE < 50
   printf("Medium array");
   #else
   printf("Big array");
   #endif
}
```

```
#define LINUX

void main()
{
   #ifdef LINUX
    printf("We are in Linux");
   #endif

   #ifdef WIN
   printf("We are in Windows");
   #endif

   #ifndef WIN
   printf("We are NOT in Windows");
   #endif
}
```

Функции

- **Функция** это фрагмент программного кода (подпрограмма), к которому можно обратиться из другого места программы по ее имени.
- После выполнения функции управление возвращается обратно в адрес возврата — точку программы, где данная функция была вызвана.
- Функция должна быть соответствующим образом *объявлена* и *определена*.
- ***Объявление функции***, кроме имени, содержит список передаваемых параметров. Параметры указанные в объявлении функции называются формальными параметрами.
- ***Определение функции*** содержит исполняемый код функции.
- Для того, чтобы использовать ранее определённую функцию, необходимо в требуемом месте программного кода указать имя функции и перечислить передаваемые в функцию параметры. Параметры указываемые в вызове функции называются фактическими параметрами.
- Функция должна быть объявлена в программе выше, чем ее вызов.
- Функция определяет собственную локальную область видимости, куда входят входные параметры, а также те переменные, которые объявляются непосредственно в теле самой функции.

Функции

- Функции в языке С состоят из 3 частей:
  1. Тип возвращаемого значение
  2. Имя функции
  3. Список параметров в круглых скобках

Вид: ВозвращаемыйТип ИмяФункции(Тип1 Параметр1, Тип2 Параметр2…)

- Если функция не принимает параметров, то после ее имени пишут пустые круглые скобки.

```
int funcNoParams()
```

- Если функция не должна возвращать значения, то в целях унификации пишут что она возвращает специальное "пустое" значение (void)

```
void someFunction(int x);
```

Если функция возвращает значение, то в коде функции пишут ключевое слово **return** для того чтобы вернуть значение вызвавшей её программе.

```
return x;
```

Функции

- Пример функции не возвращающей значения:

```
void sayHello(int n)
{
  int i;
  for (i=0; i<n; i++)
      printf(“Hello!\n”);
}
```

- Пример функции возвращающей значение:

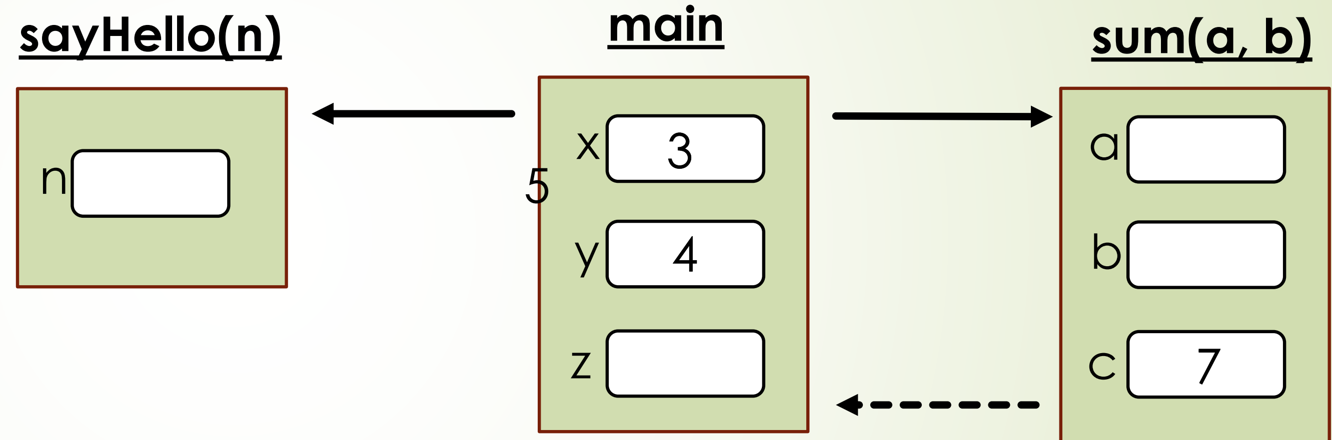
```
int sum(int a, int b)
{
      int c;
      c = a + b;
      return c;
}
```

Вызов функции и передача параметров

- Для вызова функции и исполнения её кода следует написать её имя и в круглых скобках передать фактические параметры (константы или переменные).

Пример:

```
void main()
{
   int x=3, y=4, z;
   sayHello(5);
   z = sum(x,y);
}
```



- В случае передачи в качестве параметра простого типа данных, значение параметра копируется в функцию. В случае изменения внутри функции сами исходные переменные не меняются.
- В случае передачи в функцию в качестве параметра составного типа данных (массива) передается сам массив, и в случае его изменения внутри функции эти изменения затрагивают исходный массив.

Функции с переменным числом параметров

- Язык программирования Си допускает использование функций, которые имеют переменное количество параметров.
- Функция с переменным числом параметров должна иметь как минимум один обязательный параметр.
- Для указания того, что у функции неопределенное число параметров в её объявлении используется многоточие: **тип имя_функции(обязательные параметры, ...)**
- В качестве обязательного параметра обычно указывается количество необязательных параметров.
- Для упрощения работы с функциями имеющими переменное количество параметров неопределенных типов в языке Си в стандартом заголовочном файле **stdarg.h** определены специальные макрокоманды:
  - **void va_start(va_list param, последний_явный_параметр)** - Инициализация списка необязательны параметров в переменную param типа va_list. Для этого в данную функцию нужно передать последний обязательный параметр.
  - **type va_arg(va_list param, type)** - позволяет получить из списка va_list очередной параметр типа type, а также переместить указатель va_list на следующий необязательный параметр.
  - **void va_end(va_list param)** - В качестве параметра принимает указатель va_list, который ранее был задействован в макросах va_start и va_arg. Очищает использованную под параметры память.

Функции с переменным числом параметров

- Пример функции, которая получает несколько целых чисел и возвращает их сумму

```
#include <stdarg.h>

int sumSome(int num, ...)
{
    int i, temp, sum=0;
    va_list params;
    va_start(params, num);

    for (i=0; i<num; i++)
    {
        temp = va_arg(params, int);
        sum+=temp;
    }
    va_end(params); //очищаем память
    return sum;
}
```

Основная программа

```
int main()
{
    int res;

    res = sumSome(6,6,5,4,3,2,1);
    printf("sum = %i\n", res);

    res = sumSome(7,6,5,4,3,2,1,9);
    printf("sum = %i\n", res);

    res = sumSome(3,6,3,4);
    printf("sum = %i\n", res);
}
```

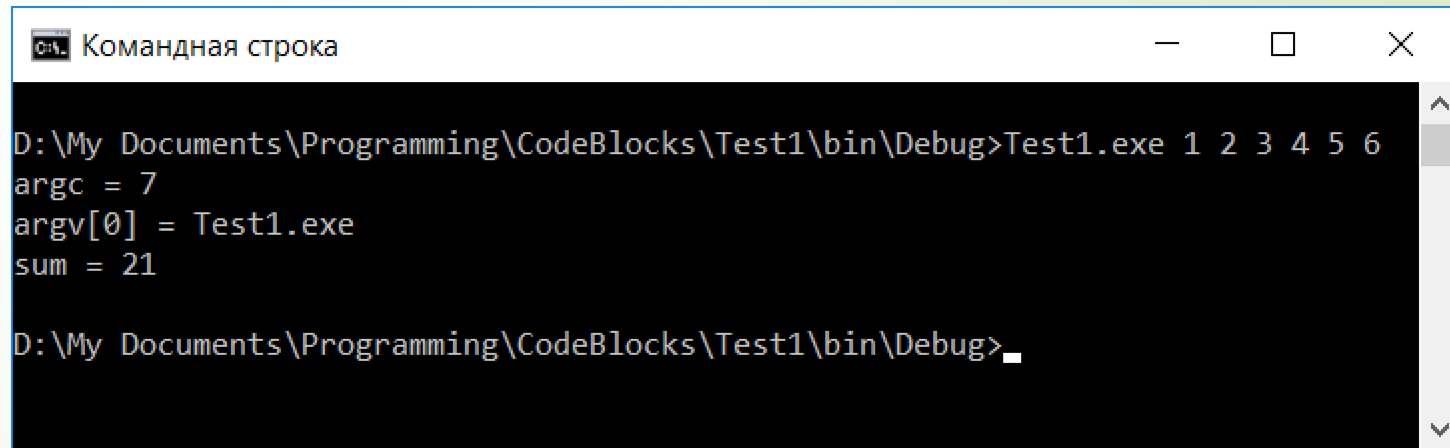
Параметры функции main()

- Консольное приложение на языке Си начинает свою работу с выполнения функции **main().**
- У функции main также могут быть параметры. Им присваиваются значения если программа вызывается из командной строки и после ее имени перечислены значения.
- Передаваемые в функцию main параметры являются текстовыми строками, которые затем можно преобразовать к нужному типу. Для преобразования строки в целое число можно использовать функцию **atoi**.
- Объявление функции main принимающей параметры, выглядит так:
  int main(int argc, char **argv)
- Параметр **argc** содержит количество параметров командной строки плюс один.
- Параметр **argv** это массив, содержащий значения переданных параметров. Нулевая ячейка массива **argv** содержит служебную информацию — полный путь и имя исполняемого файла.

Функция main с параметрами

- В IDE Code::Blocks можно задать параметры функции main в меню Project → Set programs' arguments… Program arguments.
- Пример функции main, которая получает несколько целых чисел в качестве параметров и выводит на экран их сумму.

```
int main(int argc, char **argv)
{
    int i, sum=0;

    printf("argc = %i\n", argc);
    printf("argv[0] = %s\n", argv[0]);

    for (i=1; i<argc; i++)
        sum+=atoi(argv[i]);

     printf("sum = %i\n", sum);
}
```



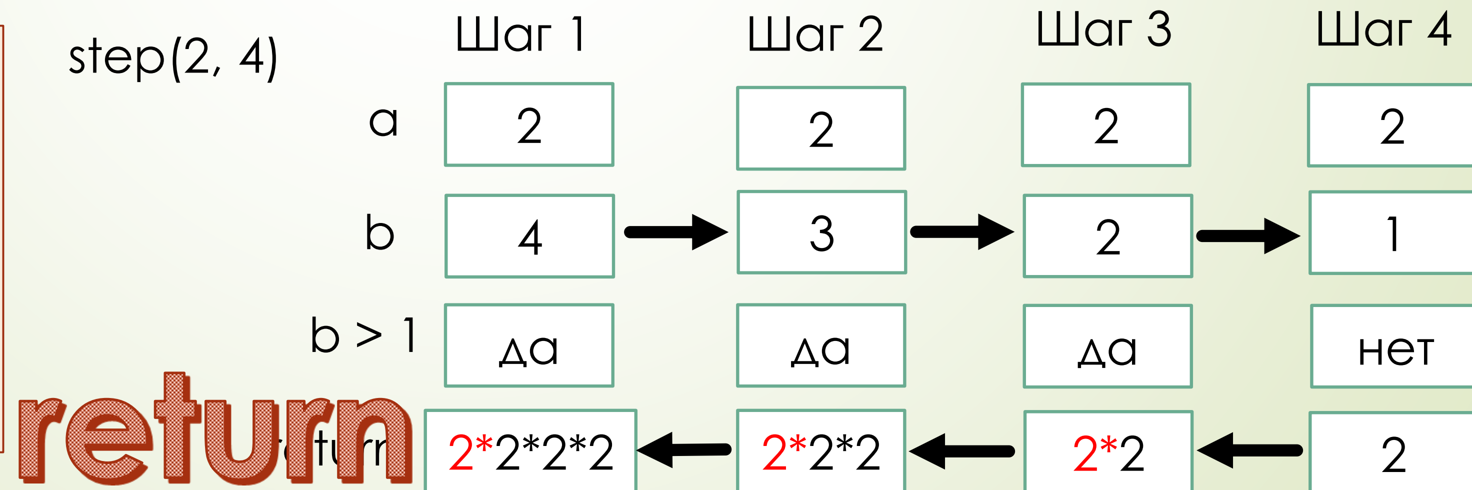
Рекурсивные функции

- Рекурсивной называется функция, которая вызывает сама себя.
- Когда функция рекурсивно обращается сама к себе, каждое следующее обращение сопровождается получением ею нового набора локальных переменных, независимых от предыдущих наборов.
- В рекурсивной функции должно быть определено условие остановки, когда функция прекращает вызывать сама себя и возвращает значение.
- С помощью рекурсивных функций удобно обрабатывать рекурсивные определения.

Пример: Функцию возведения в степень можно представить в рекурсивном виде:

$A^b = A * A^{b-1}$. Рекурсия заканчивается, когда степень становится 1: $A^1 = A$

```
int step(int a, int b)
{
   if (b>1)
      return a*step(a, b-1);
   else
      return a;
}
```



Многофайловые программы

- С помощью директивы препроцессора **#include** можно разбить программу на языке Си на несколько файлов, чтобы тематически сгруппировать вместе несколько функций.
- При разбиении кода программы на несколько файлов нужно создать как минимум три файла:
  - Заголовочный файл (с расширением .h) в который помещают объявления функций.
  - Файл исходного кода (с расширением .c) в котором содержится определения функций (их код).
  - Файл основной программы, в которой эти функции будут использоваться.

func.h

```
int sum(int, int);
```

func.c

```
int sum(int a, int b)
{
    return a + b;
}
```

main.c

```
#include "func.h"

void main()
{
  int n;
  n = sum(3,6);
}
```

Занятие 6. Темы

- Enum. Перечисляемый тип данных
- Структура. Составной тип
- Выравнивание структур
- typedef. Псевдоним типов данных и структур

Enum. Перечисляемый тип данных

- Перечисляемый тип **enum** (сокращённо он англ. enumeration) это определяемый программистом тип данных, чьё множество значений представляет собой ограниченный список идентификаторов.
- Перечисление это список целых констант, то есть имен к которым жестко привязаны целочисленные значения.
- По умолчанию первое имя в перечислении имеет значение 0, следующее — 1, и т. д.
- Если некоторое значение в перечислении было задано, то следующие продолжают прогрессию, возрастая каждый раз на единицу от предыдущего значения.
- После определения перечисляемого типа, нужно создать переменные этого типа, после чего этими переменными можно будет пользоваться как обычными переменными, присваивать значения из списка, проверять содержимое, и т.д.
- Переменная типа enum, ведет себя как целочисленная переменная. Её можно использовать в операторе switch, и на экран она выводится как целое число.

Пример использования типа enum

```
enum Days {Sunday, Monday, Thuesday, Wednesday, Thursday, Friday, Saturday};
enum Days today;

today = Wednesday;

if (today == Sunday || today == Saturday)
    printf("Today is weekend!");
else
   printf("Today is %i. I'm working", today);
```

Структура. Составной тип

- Структура это определяемый программистом тип данных, несколько переменных, возможно, различных типов, которые для удобства работы с ними сгруппированы под одним именем.
- Объявление структуры начинается с ключевого слова **struct** и содержит список объявлений переменных, заключенный в фигурные скобки. За словом struct может следовать имя, называемое тегом структуры
- Перечисленные в структуре переменные называются элементами (members). Имена элементов и тегов без каких-либо коллизий могут совпадать с именами обычных переменных, так как они всегда различимы по контексту.
- После определения структуры, нужно создать переменные этого типа, после чего этими переменными можно будет пользоваться.
- При объявлении переменной типа структура её поля можно инициализировать, так же как массив, перечислив значения элементов структуры в фигурных скобках.
- Доступ к отдельному элементу структуры осуществляется посредством оператор доступа . (точки).
- Оператор доступа к элементу структуры . (точка) соединяет имя структуры и имя элемента.

Пример структуры

```
struct Car
  {
    int number;
    char color;
    char mark[15];
  };

  struct Car myCar = {234, 'R', "BMW"};
  struct Car anotherCar;

  anotherCar.number = 12345;
  anotherCar.color = 'R';
  strcpy(anotherCar.mark, "Lada Kalina");

  printf(" Car number = %i\n Car color = %c\n Car mark = %s\n", myCar.number, myCar.color,
myCar.mark);
```

Структура Car

Тип	Имя	Размер
int	number	4 байта
char	color	1 байт
char[15]	mark	15 байт

Переменная anotherCar типа «структура Car»

Элемент	Значение
number	12345
color	R
mark	Lada Kalina

Выравнивание структур

- Чтобы обращение к элементам структуры происходило наиболее эффективным способом, компилятор размещает в памяти элементы структуры таким образом, чтобы относительный адрес каждого элемента был кратен его размеру. (т.е. элемент типа int начинается с адресов 0, 4, 8… элемент short - 0, 2, 4, 6…). Для этого, при размещении структуры в памяти выполняется следующее:
  - Перед элементом вставляется такое число пустых байт (**padding**), чтобы он сдвинулся на нужный адрес.
  - В конец структуры вставляется такое число байт, чтобы общий размер структуры был кратен размеру его самого большого элемента.

```
struct Car
{
   int number;
   char color;
   char mark[10];
};
```

0	1	2	3
i		n	t
char	c	h	
a		r	
[	1	0]	x

```
struct Car
{
   char color;
   int number;
   char mark[10];
};
```

0	1	2	3
char	x	x	x
i		n	t
c		h	a
r		[	1
0]		x	x

typedef. Псевдоним для типов данных

- В языке С есть возможность давать типам данных новые имена. Это делается с помощью команды **typedef**.

  Пример:

  typedef int Length;

  Length len, maxlen;

  имя Length становится синонимом int. С этого момента тип Length можно применять в объявлениях, в операторе приведения и т. д. точно так же, как тип int

- Возможность давать типам новые имена полезна в частности при использовании структур. В это случае не придется писать слово struct при создании переменных из структуры.

```
typedef struct
  {
    int number;
    char color;
    char mark[10];
  } Car;

 Car myCar;
```

Занятие 7. Темы

- Тип данных указатель
- Получение адреса и косвенный доступ
- Указатели и аргументы функций
- Выделение памяти под указатель. Функции malloc и free
- Адресная арифметика

Тип данных указатель

- Память компьютера представляет собой массив последовательно пронумерованных ячеек с которыми можно работать по отдельности или связными кусками.
- Указатель — это переменная, содержащая адрес переменной.
- Указателю разрешено указывать только на переменные определенного типа.
- В определении указателя записано, на переменную какого типа он указывает.
- Для определения указателя используется символ звёздочка (*)

Пример:

int pt; //указатель на переменную типа int*

Получение адреса и косвенный доступ

- Адрес переменной это какое-то целочисленное значение.
- Для того чтобы получить адрес переменной используется унарный оператор амперсанд (&) который ставится перед именем переменной и возвращает её адрес.

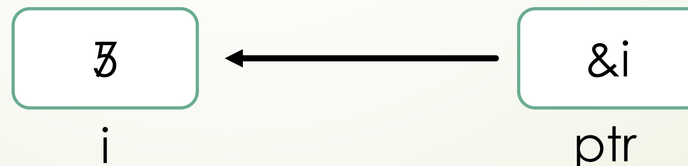
Пример:

```
int i;
int* ptr;
ptr = &i;     //в указатель ptr записывается адрес переменной i
```

- Унарный оператор звездочка (*) называется оператором косвенного доступа. Он позволяет с помощью указателя, работать с содержимым переменной, на которую тот указывает.

Пример:

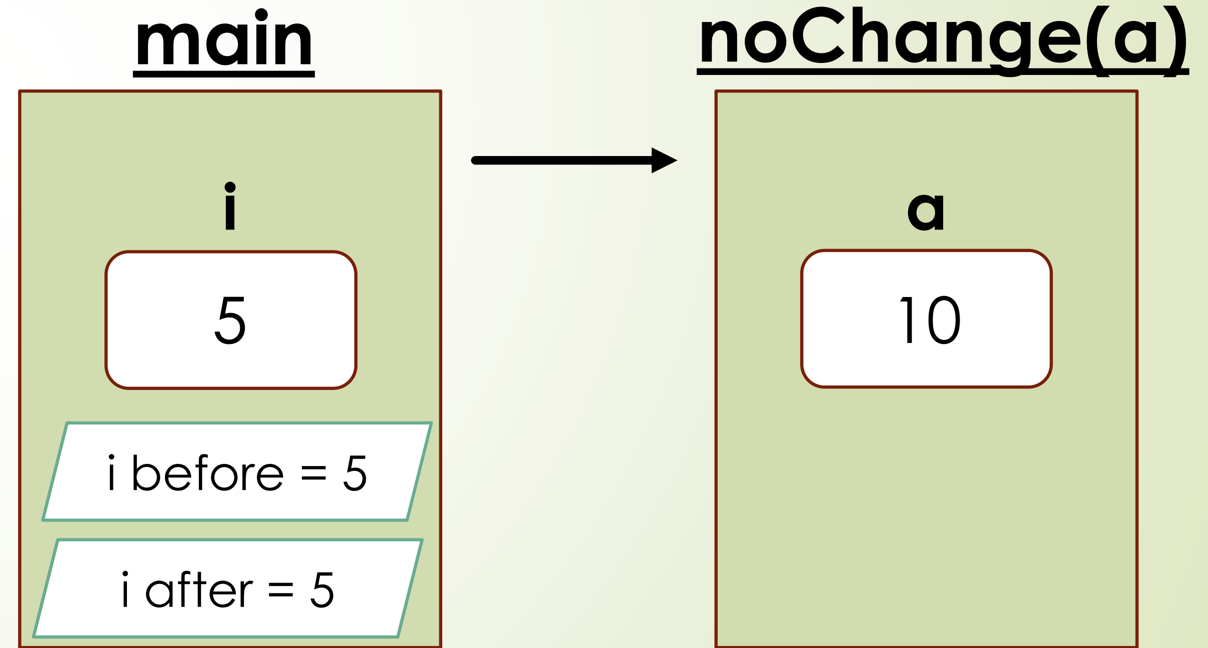
```
i = 5;
*ptr = 7;
printf("i = %i", i);
```



Указатели и аргументы функций

- В языке С функция получает в качестве параметров лишь копии переменных, а не сами переменные. Поэтому если внутри функции переменные переданные в качестве параметров меняют значения, то это никак не влияет на содержимое тех переменных которые были переданы в функцию.

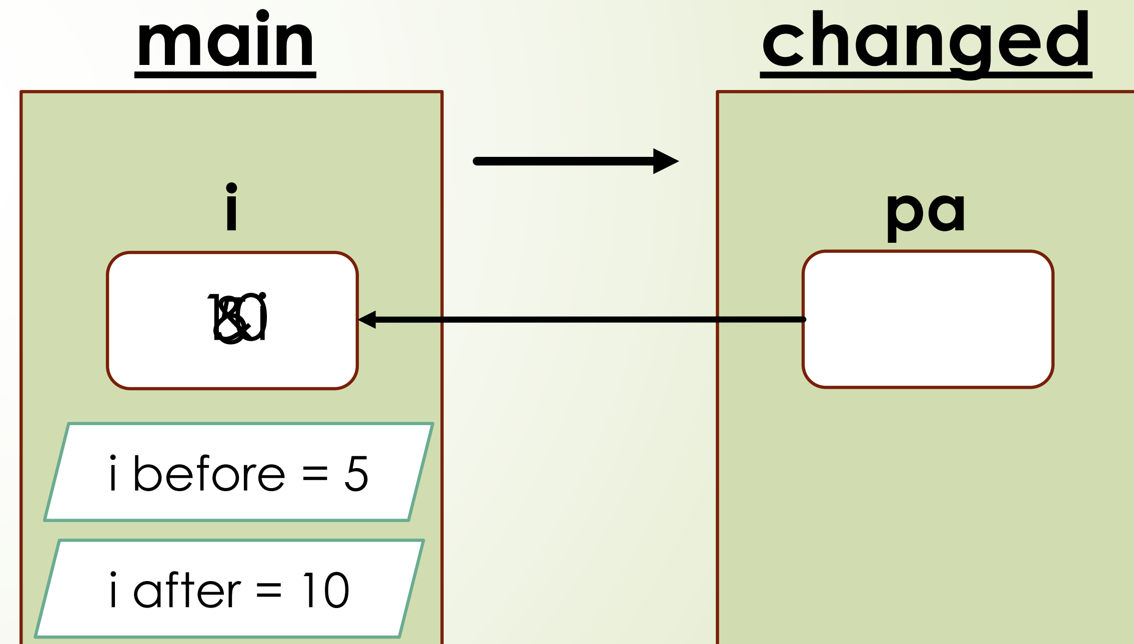
```
void noChange(int a)
{
    a = a + 5;
}

int main()
{
   int i;
   i = 5;
   printf("i before = %i\n", i);
   noChange(i);
   printf("i after = %i\n", i);
}
```



Передача в функцию адреса

- Если нужно передать в функцию переменные, значения которых следует изменить в так чтобы это изменение затронуло сами исходные значения, то следует использовать адрес и передачу указателя.

```
void changed(int *pa)  //указатель на адрес
{
   *pa = *pa + 5;
}
int main()
{
   int i;
   i = 5;
   printf("i before = %i\n", i);
   changed(&i);              //передаем адрес
   printf("i after = %i\n", i);
}
```

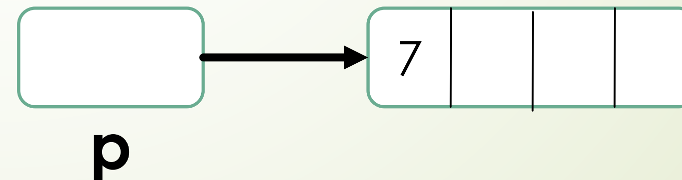


Выделение памяти

- Если мы хотим сделать так чтобы указатель указывал не на переменную, а на свободный участок памяти, произвольного размера, то нам следует специально выделить эту память функцией **malloc**.
- Функция malloc находится в библиотеке stdlib.h
- Функция malloc выделяет блок памяти, размером с такое число байт, какое указано в аргументе этой функции и возвращает указатель на начало блока.
- Указатель, который возвращает функция malloc является указателем общего вида (void*), и его нужно преобразовать к нужному нам типу указателя.
- После того, как память больше не нужна следует её освободить функцией **free**

Пример:

```
int* p;
p = (int*)malloc(4);
*p = 7;
printf("*p = %i", *p);
free(p);
```



Текст как возвращаемое значение функции

- Поскольку текст в языке С представляет собой массив символов (char), то сделать текст возвращаемым значением можно только используя указатель типа char*.
- Возвращаемое значение типа char* представляет собой адрес с которого начинается текст, поэтому использовать в качестве возвращаемого значения локально созданный массив char[] нельзя, поскольку при выходе из функции такой массив будет уничтожен и ссылаться на него будет бессмысленно.
- Также текст можно вернуть из функции не как возвращаемое значение, а как параметр.

```
//Пример. Текст возвр.значение
char* getText()
{
   //char text[20]; //нельзя!!
   char* text = malloc(20);
   strcpy(text, "Some text");

    return text;
}
```

```
//Пример. Константный текст
char* getFixedInfo()
{
    return "Fixed text";
}
```

```
//Пример. Текст как параметр
void getInfo(char info[])
{
    strcpy(text, "Some info");
}
```

```
//основная программа
void main()
{
    char* text;
    char info[20];

    text = getText();
    printf("%s\n", text);

    getInfo(info);
    printf("%s\n", info)
}
```

Адресная арифметика

- В переменной типа указатель содержится адрес какой-то ячейки памяти.
- С помощью арифметических операций над указателем можно сделать так чтобы он указывал на другую ячейку в памяти.
- Перемещать указатель имеет смысл только когда он указывает на большой блок памяти.

Выражение	Значение
***p**	***Значение*** *указуемой переменной*
p + i	*Указатель на i-ю переменную* ***после*** *указуемой*
p - i	*Указатель на i-ю переменную* ***перед*** *указуемой*
***(p+i)**	*Значение i-й переменной после указуемой*
p++	*Установить указатель на переменную, следующую за указуемой*
p--	*Установить указатель на переменную, предшествующую указуемой*
p+=i	*Переместить указатель на i переменных вперед относительно указуемой*
p-=i	*Переместить указатель на i переменных назад относительно указуемой*
***p++**	*Получить значение указуемой переменной и установить указатель на следующую*
***(--p)**	*Переместить указатель к переменной, предшествующей указуемой, и получить ее значение*

Занятие 8. Темы

- Указатели и массивы
- Одномерный динамический массив
- Двумерный динамический массив
- Указатель на функцию

Указатели и массивы

- В языке С существует тесная связь между указателями и массивами.
- С помощью адресной арифметики и указателя можно перемещаться по ячейкам памяти как по массиву.
- Доступ к элементу массива, осуществляемый обращением по индексу, может быть выполнен с помощью указателя.
- Имя массива является синонимом расположения его начального (нулевого) элемента.
- Если *pa* это указатель, то с его помощью можно работать с ячейками массива, как обращаясь к ним по индексу *pa[i]*, так и смещая указатель адресной арифметикой **(pa+i)*.
- Между именем массива *(a)* и указателем *(pa)*, выступающим в роли имени массива, существует одно различие. Указатель — это переменная, поэтому можно написать pa=a или pa++. Но имя массива не является переменной, и записи вроде a=pa или a++ не допускаются.

Одномерный динамический массив

- С помощью указателя и функций выделения памяти, можно создавать динамические массивы, размер которых становится известным только во время работы программы.
- Размер одной ячейки создаваемого массива лучше всего определять с помощью команды sizeof.
- Основные функции выделения памяти это malloc, calloc и realloc.

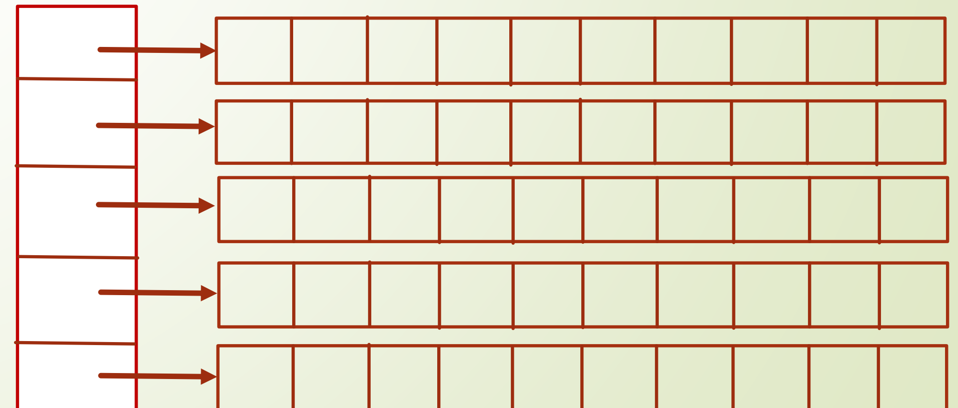
Функция	Значение
malloc(size)	Выделяет блок памяти размером size байт и возвращает адрес первой ячейки
calloc(num, size)	Выделяет память для массива из num элементов, каждый из который занимает size байт памяти, и заполняет все биты выделенной памяти нулями. Возвращает адрес первой ячейки.
realloc(ptr, newsize)	Изменяет величину выделенной памяти, на которую указывает ptr, на новую величину, задаваемую в байтах параметром newsize. Содержимое старого блока копируется в новый блок

Двумерный динамический массив

- Для работы с одномерным массивом, который содержит целочисленные значения используется указатель типа int*.
- Чтобы работать с двумерным массивом, который содержит целочисленные значения нужно создать одномерный массив каждая ячейка которого является указателем на одномерный массив, содержащий целочисленные значения.
- Чтобы создать указатель на одномерный массив, где каждая ячейка сама является указателем на массив int необходимо использовать указатель типа int**
- Выделение памяти под двумерный массив делается в два этапа:
  1. Выделить память под одномерный массив указателей (по количеству строк)
  2. В цикле выделить память для каждой строки (по количеству столбцов)

Пример:

```
int i, rows=5, cols=10;  int** matrix;
matrix = (int**)malloc(sizeof(int*)*rows);
for (i=0; i<rows; i++)
    matrix[i]= (int*)calloc(cols, sizeof(int));
```



Указатель на функцию

- В соответствии с **архитектурой фон Неймана**, по которой устроены современные компьютеры, команды и данные хранятся в одной и той же памяти и внешне неразличимы. Распознать их можно только по способу использования. Одно и то же значение в ячейке памяти может использоваться и как данные, и как команда, и как адрес в зависимости лишь от способа обращения к нему.
- Поскольку команды (функции) находятся в той же памяти что и данные, то у них есть такие же адреса, как и у данных, и с ними так же можно работать с помощью указателей.
- Определение указателя на функцию имеет следующий вид:
  тип_возвращаемого_значения (*имя_указателя) (параметры)
- Указателю на функцию можно присвоить функцию, которая соответствует указателю по возвращаемому типу и спецификации параметров.
- Используя указатели на функцию можно передавать одни функции в качестве параметра другим функциям.
- Кроме одиночных указателей на функции можно определять их массивы. Для этого используется следующий синтаксис: **тип (*имя_массива[размер]) (параметры)**

Указатель на функцию

- Пример использования указателя на функцию

Функция 1

```
int sum(int a, int b)
{
    return a + b;
}
```

Функция 2

```
int sub(int a, int b)
{
    return a - b;
}
```

Основная программа

```
int main()
{
    int s, a=3, b=9;
    int (*funcPtr)(int,int);
    funcPtr = sum;

    s = funcPtr(a, b);
    printf("sum = %i", s);

    funcOut(funcPtr, a, b);

    funcOut(sum, a, b);

    funcOut(sub, a, b);
}
```

Функция использующая указатель на функцию

```
void funcOut(int (*funcPtr)(int,int), int a, int b)
{
    int s;
    s = funcPtr(a, b);
    printf(“result from func = %i", s);
}
```

Занятие 9. Темы

- Связанные списки
- Односвязный список
- Создание односвязного списка
- Удаление элементов из списка

Динамические связанные списки

- С помощью динамического выделения памяти можно создавать массивы различных размеров и даже менять размеры массива во время работы программы. Но при вставке и удалении элементов из массива требуется много накладных расходов.
- Связанный список является более гибким средством хранения данных чем массив. В нем легко можно удалять и вставлять элементы в любое место.
- В отличие от массива связанный список является структурой данных с последовательным, а не произвольным доступом. Чтобы получить доступ к нужному элементу необходимо перебрать все элементы пока не найдется нужный.
- Односвязный список состоит из двух частей:
  1. Данные
  2. Указатель на следующий элемент списка
- Чтобы указать что элемент списка является последним и после него нет других элементов, его указателю на следующий элемент присваивается специальное значение NULL (пустота).

Односвязный список

- В односвязном списке возможен переход только вперед по списку, возврат назад не предусмотрен, поэтому необходимо всё время хранить указатель на первый элемент списка.
- Для создания связанного списка используется рекурсивная структура, одно из полей которой является указателем на такую же структуру.
- Доступ к отдельному элементу структуры через указатель осуществляется с помощью оператора косвенного доступа ->

Пример:

```
struct CarList
  {
    int number;
    char color;
    char mark[15];
    struct CarList* next;
  };
```

```
void main()
{
  struct CarList *myCar;
  myCar = (struct CarList*)malloc(sizeof(struct CarList));
  myCar->number = 12345;
  myCar->color = 'R';
  strcpy(myCar->mark, "Lada Kalina");
}
```

Пример создания односвязного списка

Структура для создания списка

```
struct LinkedList
{
      int number;
      struct LinkedList* next;
};
```

Создание элементов списка

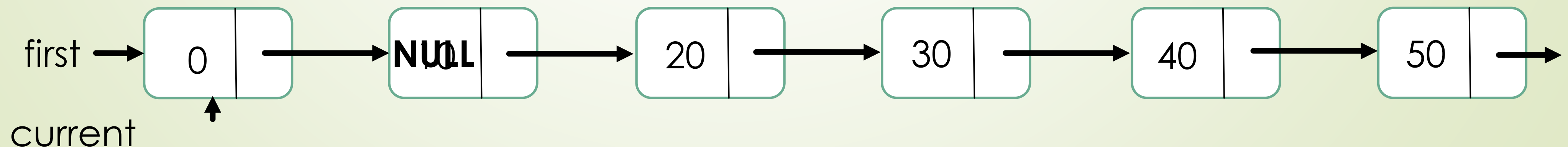
```
for (i=1;i<10; i++)
{
      current->next = malloc(sizeof(struct LinkedList));
      current=current->next;
      current->number = i*10;
      current->next = NULL;
}
```

Создание двух указателей на элементы

```
struct LinkedList *first, *current;
first = malloc(sizeof(struct LinkedList));
current = first;
first->number = 0;
first ->next = NULL;
```

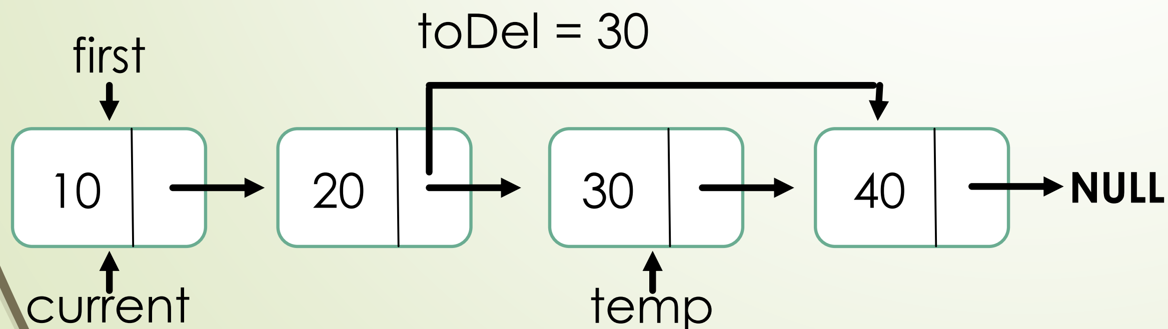
Проход по элементам списка

```
current = first; i=0;
while (current!=NULL)
{
    printf("item #%i = %i\n", i, current->number);
    current=current->next; i++;
}
```



Удаление элемента из связанного списка

- Чтобы удалить элемент из связанного списка нужно выполнить три шага:
  1. Найти нужный элемент
  2. Переставить указатели
  3. Освободить память занятую элементом
- Для освобождения памяти необходим вспомогательный указатель.
- Первый элемент в списке нужно рассмотреть отдельно, потому что если его нужно удалить то необходимо также переставить указатель на первый элемент.




```
current = first;
//если искомый элемент первый
if (current->number == toDel)
{
    current=current->next;
    free(first);
    first = current;
}
else
while (current!=NULL && current->next!=NULL)
{
    if (current->next->number == toDel)
     {
        temp = current->next;
        current->next = current->next->next;
        free(temp);
      }
   current=current->next;
}
```