Основы программирования и баз данных

Модуль 2.
ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИИ

Теория:

- Системы счисления
- Преобразования между системами счисления
- Арифметика систем счисления
- Диапазоны представления чисел
- Единицы измерения информации
- Кодировки, таблицы кодировок
- Понятие типа данных
- Выражения. Операнды. Знаки операций
- Структуры данных

Практика:

- Преобразования между системами счисления
- Сложение и вычитание в двоичной и шестнадцатеричной системах

Системы счисления.

- **Система счисления** — способ записи чисел с помощью набора специальных знаков, называемых *цифрами*.

- Системы счисления подразделяются на
  - **позиционные** (например, десятичная)
  - **непозиционные** (например, римская)

- В позиционных системах счисления величина, обозначаемая цифрой в записи числа, зависит от её положения в числе (позиции).
- Количество используемых цифр называется ***основанием системы счисления***

Материал из Википедии — свободной энциклопедии

Системы счисления (продолжение)

1) десятичная

10^3	10^2	10^1	10^0	Значение
1	1	0	9	$1*10^3+1*10^2+0*10^1+9*10^0=1109_{(10)}$

2) двоичная

2^3	2^2	2^1	2^0	Значение
1	1	0	1	$1*2^3+1*2^2+0*2^1+1*2^0=13_{(10)}$

3) восьмеричная

8^3	8^2	8^1	8^0	Значение
1	1	0	7	$1*8^3+1*8^2+0*8^1+7*8^0=583_{(10)}$

4) шестнадцатеричная

16^3	16^2	16^1	16^0	Значение
1	1	0	F	$1*16^3+1*16^2+0*16^1+15*16^0=4367_{(10)}$

Связи между системами счисления

- **Перевод из десятичной в произвольную позиционную систему счисления:**
  Для перевода необходимо делить с остатком искомое число на основание системы счисления до тех пор, пока частное больше нуля, и записать цифры всех остатков в обратном порядке.
- **Пример:**
  - **44_{10}** переведём в двоичную систему:
  - 44 делим на 2. частное 22, остаток 0
  - 22 делим на 2. частное 11, остаток 0
  - 11 делим на 2. частное 5, остаток 1
  - 5 делим на 2. частное 2, остаток 1
  - 2 делим на 2. частное 1, остаток 0
  - 1 делим на 2. частное 0, остаток 1
- Теперь, записав цифры всех остатков в обратном порядке, получим число **101100_2**

Связи между системами счисления (продолжение)

- **Перевод из двоичной в восьмеричную и шестнадцатеричную системы**

  Для этого типа операций существует упрощенный алгоритм.
- Для восьмеричной — разбиваем число на триады, преобразуем триады по таблице
- Для шестнадцатеричной — разбиваем число на тетрады, преобразуем тетрады по таблице
- Пример:
  - преобразуем **101100_2**
  - восьмеричная — 101 100 → **54_8**
  - шестнадцатеричная — 0010 1100 → **$2C_{16}$**

0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	A
1 0 1 1	B
1 1 0 0	C
1 1 0 1	D
1 1 1 0	E
1 1 1 1	F

Связи между системами счисления (продолжение)

- **Перевод из восьмеричной и шестнадцатеричной систем в двоичную**

  Для этого типа операций тоже существует упрощенный алгоритм.
- Для восьмеричной — преобразуем цифры числа по таблице в триады
- Для шестнадцатеричной — преобразуем цифры числа по таблице в тетрады
- Пример:
  - преобразуем
  - $54_8 \rightarrow$ 101 100
  - $2C_{16} \rightarrow$ 0010 1100

0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	A
1 0 1 1	B
1 1 0 0	C
1 1 0 1	D
1 1 1 0	E
1 1 1 1	F

Основы арифметики двоичных чисел

Поразрядное сложение с переносом

0	1	1	1	= ?
0	1	1	0	= ?
?	?	?	?	= ?

0	1	1	1	= 7
0	1	1	0	= 6
1	1	0	1	=13

Сдвиг влево

0	0	1	1	= ?
0	1	1	0	= ?
1	1	0	0	= ?

0	0	1	1	= 3
0	1	1	0	= 6
1	1	0	0	=12

Сдвиг вправо

1	1	0	1	= ?
0	1	1	0	= ?
0	0	1	1	= ?

1	1	0	1	= 13
0	1	1	0	= 6
0	0	1	1	= 3

2^3	2^2	2^1	2^0	Значение$_{(10)}$
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

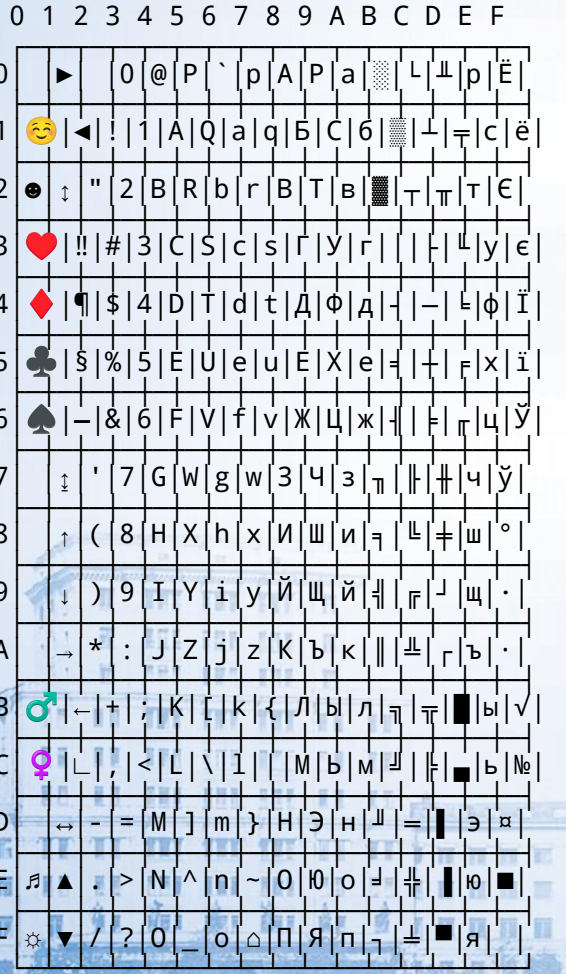
Диапазоны представления целых чисел

Кол-во бит	Запись с порядком	Обычная запись
8	$-2^7 .. 2^7-1$	-128 .. 127
16	$-2^{15} .. 2^{15}-1$	-32768 .. 32767
32	$-2^{31} .. 2^{31}-1$	-2147483648 .. 2147483647

Единицы измерения ёмкости запоминающих устройств

- 1 бит = двоичная цифра /логическое значение
- 8 бит = 1 байт - символ (ASCII)
- 1 Кбайт = 2^{10} байт
- 1 Мбайт = 2^{20} байт
- 1 Гбайт = 2^{30} байт
- 1 Тбайт = 2^{40} байт
- 1 Пбайт = 2^{50} байт **ГОСТ 8.417-2002**

Представление символьной информации. Кодовые таблицы



- Однобайтные таблицы (ASCII, ANSI, KOI-8R)
  - для представления символов используются 8-битные числовые коды
  - кодовая таблица позволяет закодировать 256 различных символов
- Двухбайтная таблица (UNICODE)
  - для представления символов используются 16-битные числовые коды
  - кодовая таблица позволяет закодировать 65536 различных символов

Практика

- **Преобразовать**:
  - 34, 65, 27, 54
  - 10110, 1100, 10101, 1010
- **Сложение и вычитание**:
  - 19 + 5 в двоичной системе счисления
  - 19 - 5 в двоичной системе счисления
- **Найти количество комбинаций 0 1:**
  - в одном байте
  - в пяти битах
- **Преобразовать документ из одной кодировки в другую.**

Понятие типа данных

Тип данных определяет:

- объем *блока памяти*, выделяемый для хранения значений:
  - 1 байт для символьного типа
  - 4 байта для целого типа
- *структурную организацию* этого блока памяти:
  - наличие или отсутствие знакового разряда для целого типа
  - наличие знакового разряда, полей порядка и мантиссы для плавающего типа
- *диапазон* возможных значений:
  - от 0 до 255 (от 00 до FF) для символьного типа
  - от -2^{n-1} до $2^{n-1}-1$ для целого типа
- набор возможных *операций*, применяемых к этим значениям:
  - для значений плавающего типа не определена операция вычисления остатка от деления
  - к значениям логического типа применяются операции отрицания, конъюнкции, дизъюнкции

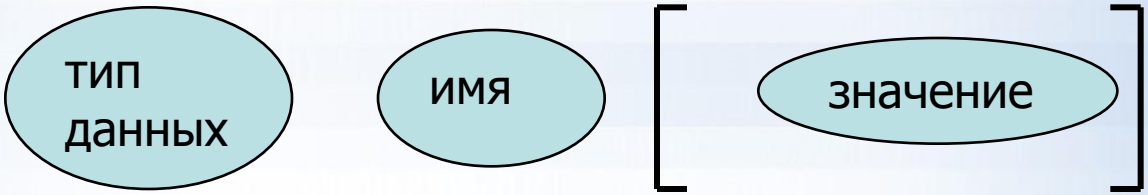
Понятие типа данных (продолжение)

В различных языках программирования реализованы те или иные из перечисленных ниже типов:

- простые (скалярные) типы:
  - логический
  - символьный
  - целый
  - с плавающей точкой
  - строковый
  - перечислимый
  - ссылочный (указатель)
- составные (структурные) типы:
  - массивы
  - записи (структуры)
  - множества
  - списки
- другие типы, определяемые программистом

Создание Переменных/Объектов

- **Что? Где? Зачем?**
- **Синтаксис:**



- **Примеры**:

```
int  a = 4
double  b
double  c = 1.57
```

Выражения. Операнды.

- **Выражение -** математическая формула или иная символическая запись, содержащая информацию о способе вычисления искомого значения.
- Синтаксически выражение строится из *операндов* и *операторов* (знаков операций).
- **Операнд** в языках программирования — аргумент операции, т.е. значение, участвующее в вычислении.
  - В зависимости от положения операнда относительно знака операции операции подразделяются на:
    - префиксные, например, $-x$,
    - инфиксные, например, $a + b$,
    - постфиксные, например, x^3.
  - В зависимости от числа операндов операции подразделяются на:
    - одноместные (унарные),
    - двуместные (бинарные),
    - многоместные операции.

Идентификаторы. Константы

- В качестве операндов в **выражениях** используются *идентификаторы*, *константы* и *другие выражения* (возможно, заключенные в скобки)
- **Идентификатор** *(символическое имя)*
  - это лексема (последовательность допустимых символов языка программирования, имеющая в нем смысл)
  - используется для именование программных сущностей (*переменных, массивов, функций* и др.)
  - делает возможным ссылки на них в тексте программы
- **Константа** (постоянная величина) — некоторая величина, не изменяющая своего значения в рамках рассматриваемого процесса.
  - *Численные литералы* (например, 0, -1 или 3.14159) всегда являются константами.
- Вычисление выражений выполняется в соответствии с **приоритетами** и **ассоциативностью** операторов (операций)

Арифметические операции

- Изменение знака **-**
- Умножение *****
- Деление **/**
- Остаток отделение **%**
- Сложение **+**
- Вычитание **-**

Операции сравнения

- Равно **==**
- Не равно **!=**
- Меньше **<**
- Меньше или равно **<=**
- Больше **>**
- Больше или равно **>=**

Результат всех операций сравнения: **True / False**

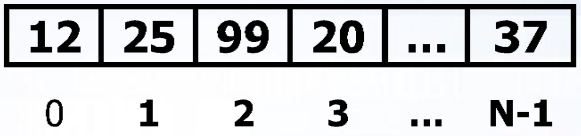
Логические операции

- Отрицание **not**
- Логическое умножение(конъюнкция) **and**
- Логическое сложение (дизъюнкция) **or**

A	B		not A	A and B	A or B
0	0		1	0	0
1	0		0	0	1
0	1			0	1
1	1			1	1

Базовые структуры данных – массивы

- **Массив (индексный)** — простая статическая структура данных, предназначенная для хранения набора единиц данных, каждая из которых идентифицируется индексом или набором индексов.
- **Индекс** — обычно целое число, либо значение типа, приводимого к целому, указывающее на конкретный элемент массива.



- Количество используемых индексов массива может быть различным. Массивы с одним индексом называют *одномерными*, с двумя — *двумерными* и т. д.
- Одномерный массив соответствует вектору в математике, двумерный -- матрице.

Материал из Википедии — свободной энциклопедии

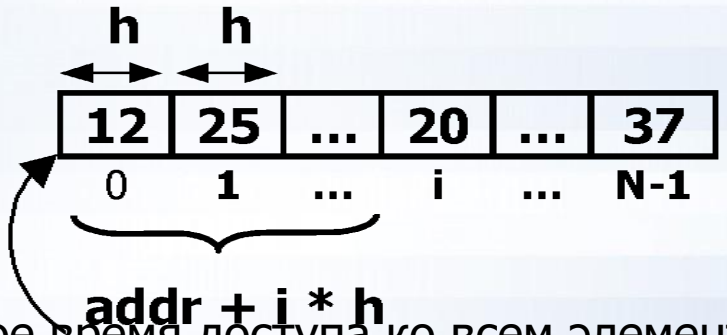
Базовые структуры данных – массивы (продолжение)

- **Специфические типы массивов**
  - **Статические массивы.**
    - Статическим называется массив, размер которого не может изменяться во время исполнения программы
  - **Динамические массивы.**
    - ***Динамическим*** называется массив, размер которого может меняться во время исполнения программы.
    - Для реализации динамики язык программирования должен предоставлять встроенную поддержку.
  - **Гетерогенные массивы.**
    - Гетерогенным называется массив, элементы которого могут содержать значения, относящиеся к различным типам данных.
    - Гетерогенные массивы удобны как универсальная структура для хранения наборов данных произвольных типов, но требуют усложнения механизма поддержки массивов в трансляторе языка.

Материал из Википедии — свободной энциклопедии

Базовые структуры данных – массивы (продолжение)

- **Достоинства массивов**
  - легкость вычисления адреса элемента по его индексу (поскольку элементы массива располагаются один за другим)



  - одинаковое время доступа ко всем элементам
  - малый размер элементов: они состоят только из информационного поля

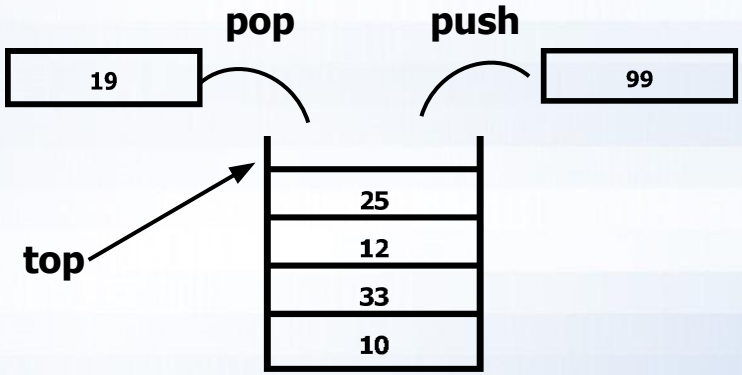
Базовые структуры данных – массивы (продолжение)

- **Недостатки массивов**
  - для статического массива
    - отсутствие динамики — невозможность удаления или добавления элемента без сдвига других
  - для динамического и/или гетерогенного массива
    - более низкое (по сравнению с обычным статическим) быстродействие
    - дополнительные накладные расходы на поддержку динамических свойств и/или гетерогенности

Материал из Википедии — свободной энциклопедии

Динамические структуры данных. Стек.

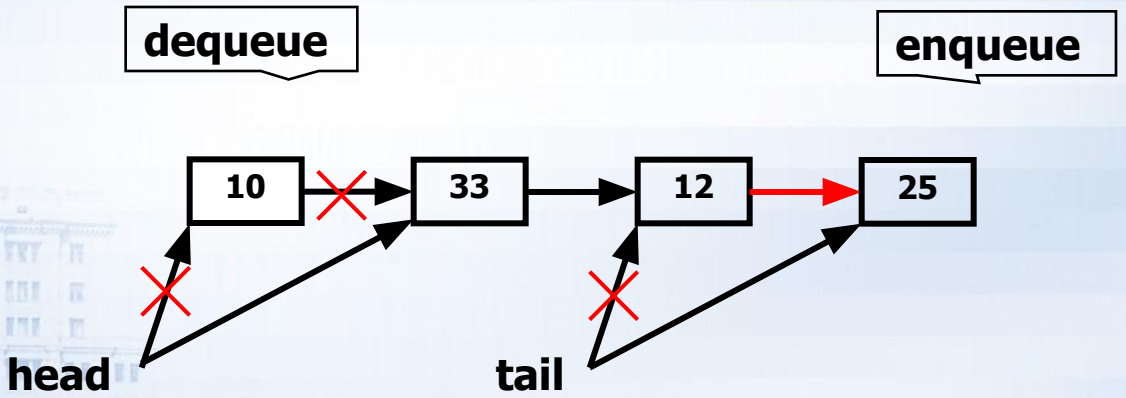
- **Стек** (англ. *stack* = стопка) — структура хранения данных с дисциплиной доступа к элементам ***LIFO*** (*Last In — First Out*, «последним пришёл — первым вышел»).
  - Операцию добавления элемента на вершину ***(top)*** стека называют ***push***, извлечения — ***pop***.



- Стек широко используется в программировании и является неотъемлемой частью архитектуры современных процессоров.
  - Компиляторы языков программирования высокого уровня используют стек для передачи параметров при вызове подпрограмм, процессоры — для хранения адреса возврата из подпрограмм.

Динамические структуры данных. Очередь.

- **Очередь** (англ. *queue*) — структура хранения данных с дисциплиной доступа к элементам ***FIFO*** (*First In — First Out,* "первый пришел - первый вышел").
  - Добавление элемента возможно лишь в конец ***(tail)*** очереди, выборка - только из начала ***(head)*** очереди, при этом выбранный элемент из очереди удаляется.
  - Операцию добавление элемента называют ***enqueue,*** выборки - ***dequeue***.



- Очередь широко используется в программировании для синхронизации процессов обработки (например, сообщений), моделирования систем массового обслуживания и т.д.

Динамические структуры данных. Список.

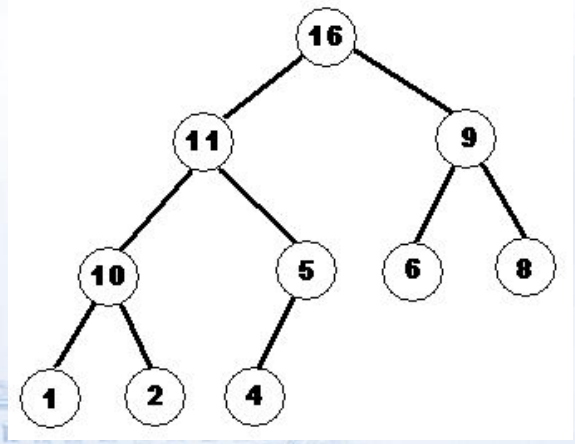
- **Связанный список** — динамическая структура хранения данных, каждый элемент которой состоит из:
  - информационного поля (содержит значение элемента)
  - одного (*односвязный*) или двух (*двусвязный*) указателей на соседние элементы.

- Список может быть *сортированным* или *несортированным*
- **Достоинства**
  - лёгкость добавления и удаления элементов
  - размер списка ограничен только объемом доступной памяти
- **Недостатки**
  - сложно определить адрес элемента по его индексу (номеру) в списке
  - на поле указателей расходуется дополнительная память (в массивах указатели не нужны)

Материал из Википедии — свободной энциклопедии

Динамические структуры данных. Деревья

- **Двоичное дерево** — абстрактная структура данных, являющееся программной реализацией двоичного дерева (графа).
- Дерево состоит из узлов (записей) вида
  (***data, l, r***),
  - где ***data*** — некоторые данные привязанные к узлу,
  - ***l***, ***r*** — ссылки на узлы, являющиеся потомками данного узла.
    Узел ***l*** называется левым потомком, а узел ***r*** — правым.



Материал из Википедии — свободной энциклопедии

Основные операции над структурами данных

- Создать (пустую) структуру данных
- Добавить новый элемент
- Удалить заданный элемент
- Проверить структуру на наличие в ней элементов
- Найти элемент(ы) с заданными свойствами
- Извлечь значение заданного элемента
- Получить элемент, следующий (в некотором порядке) за текущим
- Перебрать (в некотором порядке) все элементы структуры данных
- Сортировать (в некотором порядке) все элементы структуры данных
- Удалить всю структуру данных