Программирование и основы алгоритмизации

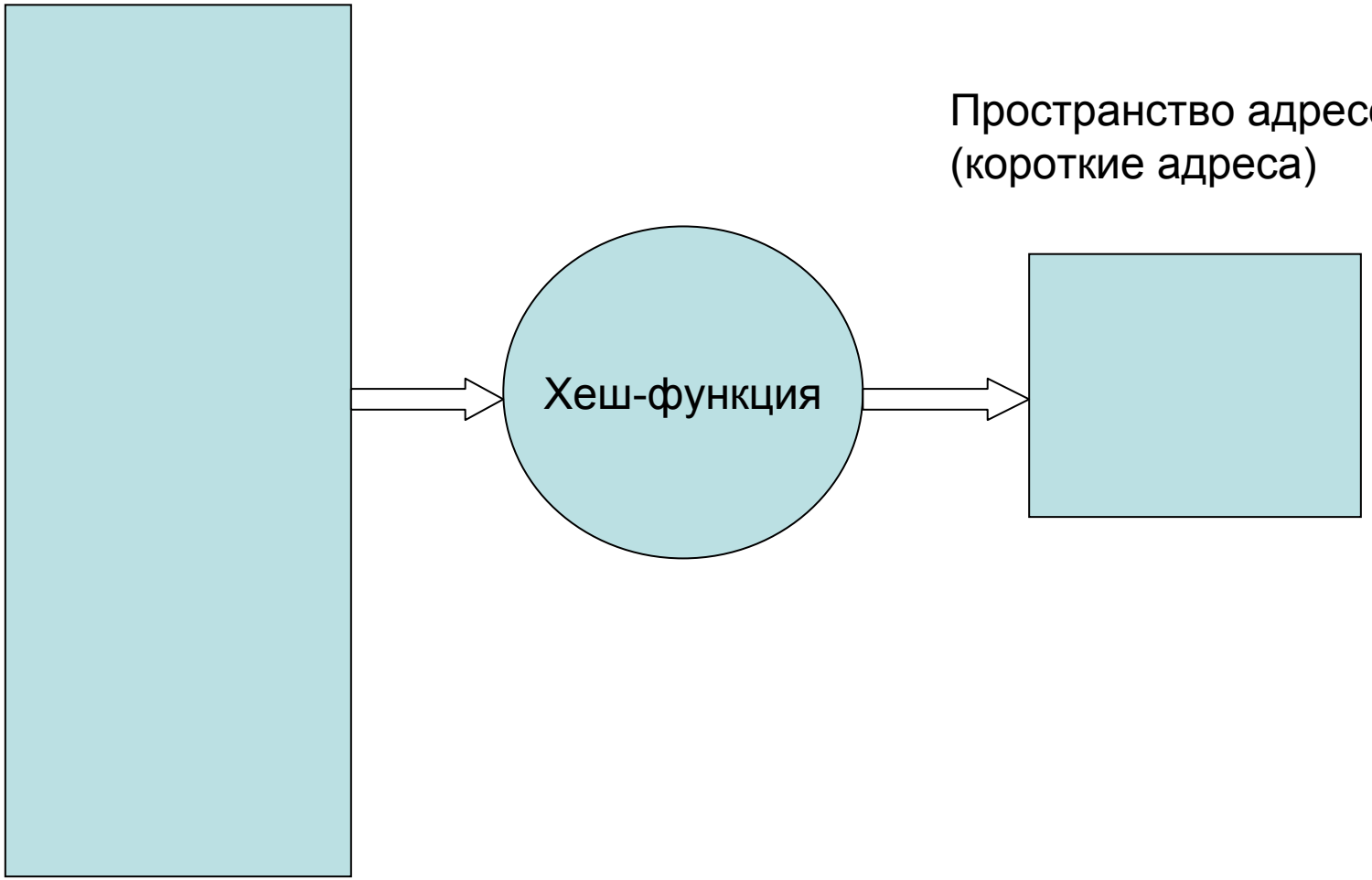
Лекция 6

Хэш-таблицы

Идея

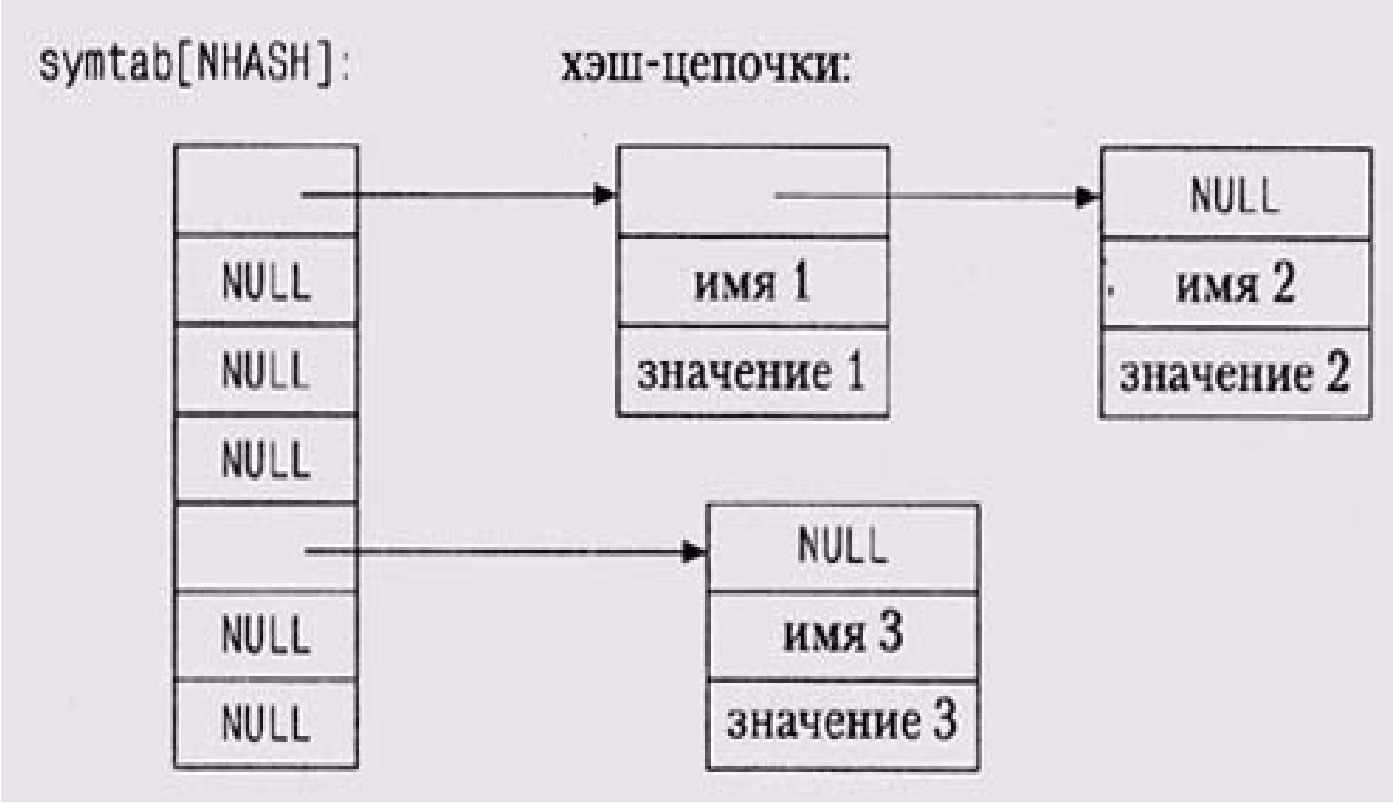
- Сочетание массивов и списков с небольшой добавкой математики позволило создать эффективную структуру для хранения и получения динамических данных
- Идея состоит в том, чтобы пропустить ключ через хэш-функцию (hash function) для получения хэш-значения (hash value), которое было бы равномерно распределено по диапазону целых чисел приемлемого размера
- Это хэш-значение используется как индекс в таблице, где хранится информация
- Java предоставляет стандартный интерфейс к хэш-таблицам
- В C и C++ обычно с каждым хэш-значением (или "bucket" - "корзиной") ассоциируется список элементов, которые обладают этим значением, как показано на следующем рисунке

Пространство имен
(длинные имена)



Пространство адресов
(короткие адреса)

Цепочки (списки) используются т.к. каждому короткому адресу может соответствовать несколько разных имен



Применения хэш-таблиц

- Типичное применение хэш-таблиц -символьная таблица, которая ассоциирует некоторое значение (данные) с каждым членом динамического набора строк (ключей)
- Компилятор практически наверняка использует хэш-таблицу для управления информацией о переменных в вашей программе
- WEB-браузер наверняка использует хэш-таблицу для хранения адресов страниц, которые вы недавно посещали,
- При соединении вашего компьютера с Интернетом, вероятно, она применяется для оперативного хранения (cache — кэширования) недавно использованных доменных имен и их IP-адресов.

Поиск имени в хэш-таблице

```
/* lookup: поиск имени в таблице; возможна вставка */
Nameval* lookup(char *name, int create, int value)
{
    int h;
    Nameval *sym;

    h = hash(name);
    for (sym = symtab[h]; sym != NULL; sym = sym->next)
        if (strcmp(name, sym->name) == 0)
            return sym;
    if (create) {
        sym = (Nameval *) emalloc(sizeof(Nameval));
        sym->name = name;          /* считаем, что память
                                      уже выделена */
        sym->value = value;
        sym->next = symtab[h];
        symtab[h] = sym;
    }
    return sym;
}
```

Пример хэш-функции

- Она должна быть:
  - детерминированной,
  - достаточно быстрой и
  - распределять данные по массиву равномерно.

  Один из наиболее распространенных алгоритмов хэширования для строк получает хэш-значение, добавляя каждый байт строки к произведению предыдущего значения на некий фиксированный множитель (хэш).

  Умножение распределяет биты из нового байта по всему до сих пор не считанному значению, так что в конце цикла мы получим хорошую смесь входных байтов. Эмпирически установлено, что значения 31 и 37 являются хорошими множителями в хэш-функции для строк ASCII.

Пример хэш-функции (2)

```
enum { MULTIPLIER = 31 };

/* hash: вычислить хэш-функцию строки */
unsigned int hash(char *str)
{
    unsigned int h;
    unsigned char *p;
    h = 0;
    for (p = (unsigned char *) str; *p != '\0'; p++)
        h = MULTIPLIER * h + *p;
    return h % NHASH;
}
```