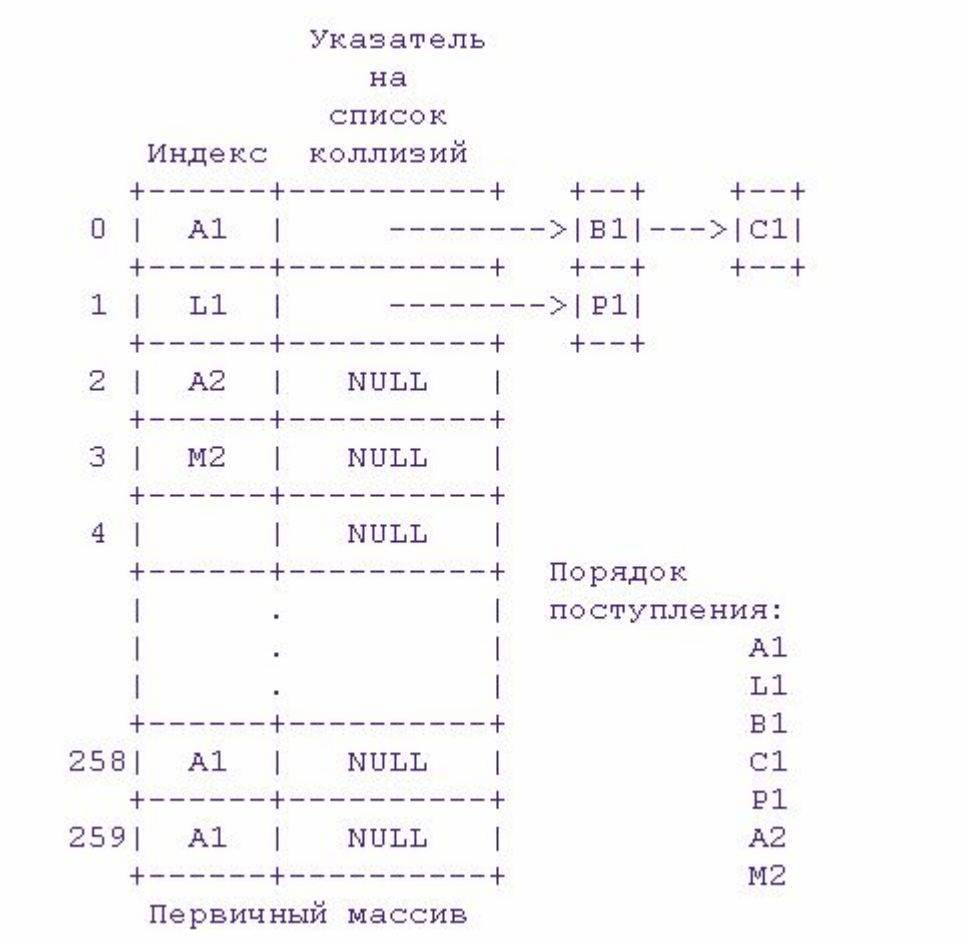
Хэширование

Хэширование (hashing) — это процесс получения индекса элемента массива непосредственно в результате операций, производимых над ключом, который хранится вместе с элементом или даже совпадает с ним. Генерируемый индекс называется хэш-адресом (hash). Традиционно хэширование применяется к дисковым файлам как одно из средств уменьшения времени доступа. Тем не менее, этот общий метод можно применить и с целью доступа к разреженным массивам. В предыдущем примере с массивом указателей использовалась специальная форма хэширования, которая называется прямая адресация. В ней каждый ключ соответствует одной и только одной ячейке массива. Другими словами, каждый индекс, вычисленный в результате хэширования, уникальный. (При представлении разреженного массива в виде массива указателей хэш-функция не должна обязательно реализовывать прямую адресацию — просто это был очевидный подход к реализации электронной таблицы.) В реальной жизни схемы прямого хэширования встречаются редко; обычно требуется более гибкий метод.

В примере с электронной таблицей понятно, что даже в самых сложных случаях используются не все ячейки таблицы. Предположим, что почти во всех случаях фактически занятые ячейки составляют не более 10 процентов потенциально доступных мест. Это значит, что если таблица имеет размер 260x100 (2`600 ячеек), в любой момент времени будет использоваться лишь примерно 260 ячеек. Этим подразумевается, что самый большой массив, который понадобится для хранения всех занятых ячеек, будет в обычных условиях состоять только из 260 элементов. Но как ячейки логического массива сопоставить этому меньшему физическому массиву? И что происходит, когда этот массив переполняется? Ниже предлагается одно из возможных решений.

Когда пользователь вводит данные в ячейку электронной таблицы (т.е. заполняет элемент логического массива), позиция ячейки, определяемая по ее имени, используется для получения индекса (хэш-адреса) в меньшем физическом массиве. При выполнении хэширования физический массив называется также первичным массивом. Индекс в первичном массиве получается из имени ячейки, которое преобразуется в число, точно так, как и в примере с массивом указателей. Но затем это число делится на 10, в результате чего получается начальная точка входа в первичный массив. (Помните, что в данном случае размер физического массива составляет только 10 % размера логического массива.) Если ячейка физического массива по этому индексу свободна, в нее заносятся логический индекс и данные. Но поскольку 10 логических позиций

соответствуют одной физической позиции, могут возникнуть коллизии при вычислении хэш-адресов. Когда это происходит, записи сохраняются в связанном списке, иногда называемом списком коллизий (collision list). С каждой ячейкой первичного массива связан отдельный список коллизий. Конечно, до возникновения коллизии эти списки имеют нулевую длину, как показано на рисунке



Предположим, требуется найти элемент в физическом массиве по его логическому индексу. Сначала необходимо преобразовать логический индекс в соответствующее значение хэш-адреса и проверить, соответствует ли логический индекс, хранящийся в полученной позиции физического массива, искомому. Если да, информацию можно извлечь. В противном случае необходимо просматривать список коллизий для данной позиции до тех пор, пока не будет найден требуемый логический индекс или пока не будет достигнут конец цепочки.

В примере хэширования используется массив структур под названием primary:

```
#define MAX 260

struct htype {
  int index;   /* логический индекс */
  int val;     /* собственно значение элемента данных */
  struct htype *next; /* указатель на следующий элемент с таким же
                         хэш-адресом */
} primary[MAX];
```

Перед использованием этого массива необходимо его инициализировать. Следующая функция присваивает полю index значение -1 (значение, которое по определению никогда не будет сгенерировано в качестве индекса); это значение обозначает пустой элемент. Значение NULL в поле next соответствует пустой цепочке хэширования.

```
/* Инициализация хэш-массива. */
void init(void)
{
  register int i;

  for (i=0; i<MAX; i++) {
    primary[i].index = -1;
    primary[i].next = NULL;  /* пустая цепочка */
    primary[i].val = 0;
  }
}
```

Процедура store() преобразует имя ячейки в хэш-адрес в первичном массиве primary. Если позиция, на которую указывает значение хэш-адрес, занята, процедура автоматически добавляет запись в список коллизий с помощью модифицированной версии функции slstore() из предыдущей главы. Логический индекс также сохраняется, поскольку он понадобится при извлечении элемента. Данные функции показаны ниже:

```
/* Вычисление хэш-адреса и сохранение значения. */
void store(char *cell_name, int v)
{
  int h, loc;
  struct htype *p;
  /* Получение хэш-адреса */
  loc = *cell_name - 'A'; /* столбец */
  loc += (atoi(&cell_name[1])-1) * 26; /* строка * ширина + столбец */
  h = loc/10; /* hash */
  /* Сохранить в полученной позиции, если она не занята
     либо если логические индексы совпадают - то есть, при обновлении.
  */
  if(primary[h].index==-1 || primary[h].index==loc) {
    primary[h].index = loc;
    primary[h].val = v;
    return;
  }
```

```
/* в противном случае, создать список коллизий
     либо добавить в его элемент */
  p = (struct htype *) malloc(sizeof(struct htype));
  if(!p) {
    printf("Не хватает памяти\n");
    return;
  }
  p->index = loc;
  p->val = v;
  slstore(p, &primary[h]);
}
/* Добавление элементов в список коллизий. */
void slstore(struct htype *i,
             struct htype *start)
{
  struct htype *old, *p;
  old = start;
  /* найти конец списка */
  while(start) {
    old = start;
    start = start->next;
  }
  /* связать с новой записью */
  old->next = i;
  i->next = NULL;
}
```

Для того чтобы получить значение элемента, программа должна сначала вычислить хэш-адрес и проверить, совпадает ли с искомым логический индекс, хранящийся в полученной позиции физического массива. Если совпадает, возвращается значение ячейки; в противном случае — производится поиск в списке коллизий. Функция find(), выполняющая эти задачи, показана ниже:

```
/* Вычисление хэш-адреса и получение значения. */
int find(char *cell_name)
{
  int h, loc;
  struct htype *p;

  /* получение значения хэш-адреса */
  loc = *cell_name - 'A'; /* столбец */
  loc += (atoi(&cell_name[1])-1) * 26; /* строка * ширина + столбец */
  h = loc/10;

  /* вернуть значение, если ячейка найдена */
  if(primary[h].index == loc) return(primary[h].val);
  else { /* в противном случае просмотреть список коллизий */
```

```
p = primary[h].next;
    while(p) {
      if(p->index == loc) return p->val;
      p = p->next;
    }
    printf("Ячейки нет в массиве\n");
    return -1;
  }
}
```

Показанный выше алгоритм хэширования очень прост. Как правило, на практике применяются более сложные методы, обеспечивающие более равномерное распределение индексов в первичном массиве, что устраняет длинные цепочки хэширования. Тем не менее, основной принцип остается таким же.