Динамически расширяемые массивы

Функция realloc

```
void* realloc(void *ptr, size_t size);
```

- ptr == NULL && size != 0

  Выделение памяти (как malloc)

- ptr != NULL && size == 0
  - Освобождение памяти аналогично free().

- ptr != NULL && size != 0

  Перевыделение памяти. В худшем случае:
  - выделить новую область
  - скопировать данные из старой области в новую
  - освободить старую область

Ошибки при использовании realloc

Неправильно

```
int *p = malloc(10 * sizeof(int));

p = realloc(p, 20 * sizeof(int));
// А если realloc вернула NULL?
```

Правильно

```
int *p = malloc(10 * sizeof(int)), *tmp;

tmp = realloc(p, 20 * sizeof(int));
if (tmp)
    p = tmp;
else
    // обработка ошибки
```

Ошибки при использовании realloc

```
int* select_positive(const int *a, int n, int *k)
{
    int m = 0;
    int *p = NULL;

    for (int i = 0; i < n; i++)
        if (a[i] > 0)
        {
            m++;
            p = realloc(p, m * sizeof(int));
            p[m-1] = a[i];
        }

    *k = m;
    return p;
}
```

Динамически расширяемые массивы

- Для уменьшение потерь при распределении памяти изменение размера должно происходить относительно крупными блоками.
- Для простоты реализации указатель на выделенную память должен храниться вместе со всей информацией, необходимой для управления динамическим массивом.

Динамически расширяемый массив

```
struct dyn_array
{
    int len;
    int allocated;
    int step;
    int *data;
};

#define INIT_SIZE 1

void init_dyn_array(struct dyn_array *d)
{
    d->len = 0;
    d->allocated = 0;
    d->step = 2;
    d->data = NULL;
}
```

Добавление элемента

```
int append(struct dyn_array *d, int item)
{
    if (!d->data)
    {
        d->data = malloc(INIT_SIZE * sizeof(int));
        if (!d->data)
            return -1;
        d->allocated = INIT_SIZE;
    }
    else
        if (d->len >= d->allocated)
        {
            int *tmp = realloc(d->data,
                        d->allocated * d->step * sizeof(int));
            if (!tmp)
                return -1;
            d->data = tmp;
            d->allocated *= d->step;
        }
    d->data[d->len] = item;
    d->len++;
    return 0;
}
```

Динамически расширяемые массивы: особенности реализации

- Удвоение размера массива при каждом вызове realloc сохраняет средние «ожидаемые» затраты на копирование элемента.
- Поскольку адрес массива может измениться, программа должна обращаться к элементами массива по индексам.
- Благодаря маленькому начальному размеру массива, программа сразу же «проверяет» код, реализующий выделение памяти.

Удаление элемента

```
int delete(struct dyn_array *d, int index)
{
    if (index < 0 || index >= d->len)
        return -1;

    memmove(d->data + index, d->data + index + 1,
                        (d->len - index - 1) * sizeof(int));
    d->len--;

    return 0;
}
```

Удаление элемента: на что обратить внимание

- Важен ли порядок элементов в массиве?
  - Нет: на место удаляемого записать последний.
  - Да: сдвинуть элементы за удаляемым вперед.
- for, memcpy или memmove?
  - for
  - memcpy НЕЛЬЗЯ (как и strcpy), memmove надежнее.
- А нужно ли удалять элементы?

Достоинства и недостатки массивов

«+»

- Простота использования.
- Константное время доступа к любому элементу.
- Не тратят лишние ресурсы.
- Хорошо сочетаются с двоичным поиском.

«-»

- Хранение меняющегося набора значений.