Основы алгоритмизации и программирования

Лекция 10

Работа с динамической памятью

Адресная функция

Векторная память поддерживается почти всеми языками высокого уровня и предназначена для хранения массивов различной размерности и различных размеров. Каждому массиву выделяется непрерывный участок памяти указанного размера.

Элементы, например, двухмерного массива ***X*** размерностью ***n1×n2*** размещаются в оперативной памяти в следующей последовательности:
***X*(0,0), *X*(0,1), *X*(0,2),... *X*(0, *n2*–1), ..., *X*(1,0), *X*(1,1), *X*(1,2),... *X*(1, *n2*–1), ..., *X*(*n1*–1,0), *X*(*n1*–1,1), *X*(*n1*–1,2), ..., *X*(*n1*–1, *n2*–1).**

Адресация элементов массива определяется некоторой адресной функцией, связывающей адрес и индексы элемента.
Пример адресной функции для массива ***X***:

$$K(i, j) = n2*i + j;$$

где $i = 0,1,2,\ldots,(n1-1)$; $j = 0,1,2,\ldots,(n2-1)$;
j – изменяется в первую очередь

Адресная функция двухмерного массива ***A*(*n*,*m*)** будет выглядеть так:

$$N1 = K(i, j) = m*i + j,$$

$i=0,1,\ldots, n-1$; $j=0,1,\ldots, m-1$.

Адресная функция

Тогда справедливо следующее:

$$A(i, j) \leftrightarrow B(K(i, j)) = B(N1),$$

B – одномерный массив с размером $N1 = n*m$.

Например, для двухмерного массива *A*(2,3) имеем:

(0,0)	(0,1)	(0,2)	(1,0)	(1,1)	(1,2)	– индексы массива A;
0	1	2	3	4	5	– индексы массива B.

Проведем расчеты:

i = 0, *j* = 0 *N*1 = 3*0+0 = 0 *B*(0)

i = 0, *j* = 1 *N*1 = 3*0+1 = 1 *B*(1)

i = 0, *j* = 2 *N*1 = 3*0+2 = 2 *B*(2)

i = 1, *j* = 0 *N*1 = 3*1+0 = 3 *B*(3)

i = 1, *j* = 1 *N*1 = 3*1+1 = 4 *B*(4)

i = 1, *j* = 2 *N*1 = 3*1+2 = 5 *B*(5)

Адресная функция

Аналогично получаем адресную функцию для трехмерного массива ***X*(*n*1, *n*2, *n*3)**:

$$K(i, j, k) = n3*n2*i + n2*j + k ,$$

где $i = 0,1,2,\ldots ,(n1-1)$; $j = 0,1,2,\ldots ,(n2-1)$;); $k = 0,1,2,\ldots ,(n3-1)$;

значение *k* – изменяется в первую очередь.

Для размещения такого массива потребуется участок оперативной памяти размером **(*n*1**n*2**n*3)****sizeof*(*type*)**. Рассматривая такую область как одномерный массив ***Y*(0,1,..., *n*1**n*2**n*3),** можно установить соответствие между элементом трехмерного массива ***X*** и элементом одномерного массива ***Y***:

$$X(i, j, k) \leftrightarrow Y(K(i, j, k)) .$$

Необходимость введения адресных функций возникает лишь в случаях, когда требуется изменить способ отображения с учетом особенностей конкретной задачи.

Работа с динамической памятью

Указатели чаще всего используют при работе с динамической памятью, которую иногда называют «***куча***» (перевод английского слова ***heap***). Это **свободная память**, в которой можно во время выполнения программы выделять место в соответствии с потребностями. **Доступ к выделенным участкам динамической памяти производится только через указатели**. Время жизни динамических объектов – от точки создания до конца программы или до явного освобождения памяти.

Некоторые задачи исключают использование структур данных фиксированного размера и требуют введения структур динамических, способных **увеличивать или уменьшать свой размер** уже в **процессе работы программы**. Основу таких структур составляют **динамические переменные**.

Динамическая переменная хранится в некоторой области оперативной памяти, не обозначенной именем, и обращение к ней производится через **переменную-указатель**.

Библиотечные функции

Функции для манипулирования динамической памятью в стандарте **Си**

void *calloc*(*unsigned n, unsigned size*);** – выделение памяти для размещения ***n объектов размером ***size*** байт и заполнение полученной области нулями; возвращает указатель на выделенную область памяти;

void *malloc* (*unsigned n*)** – выделение области памяти для размещения блока размером ***n байт; возвращает указатель на выделенную область памяти;

void *realloc* (*void *b, unsigned n*)** – изменение размера размещенного по адресу ***b блока на новое значение ***n*** и копирование (при необходимости) содержимого блока; возвращает указатель на перераспределенную область памяти; при возникновении ошибки, например, нехватке памяти, эти функции возвращают значение ***NULL***, что означает отсутствие адреса (нулевой адрес);

void free* (*void *b*)** – освобождение блока памяти, адресуемого указателем ***b

Для использования этих функций требуется подключить к программе в зависимости от среды программирования заголовочный файл *alloc.h* или *malloc.h*.

Динамический массив

В языке **Си** размерность массива при объявлении должна задаваться константным выражением.
Если до выполнения программы неизвестно, сколько понадобится элементов массива, нужно использовать **динамические массивы**, т.е. при необходимости работы с массивами переменной размерности вместо массива достаточно объявить указатель требуемого типа и присвоить ему адрес свободной области памяти (**захватить память**).

Память под такие массивы выделяется с помощью функций ***malloc*** и ***calloc*** во время выполнения программы. Адрес начала массива хранится в переменной-указателе

Пример

```
int n = 10;
double *b = (double *) malloc(n * sizeof (double));
```

Обнуления памяти при ее выделении не происходит. Инициализировать динамический массив при декларации нельзя.

Динамический массив

Обращение к элементу динамического массива осуществляется так же, как и к элементу обычного – например ***a*[3]**. Можно обратиться к элементу массива и через косвенную адресацию – ***(*a* + 3)**. В любом случае происходят те же действия, которые выполняются при обращении к элементу массива, декларированного обычным образом.

После работы захваченную под динамический массив память необходимо освободить, для нашего примера

***free*(*b*);**

Таким образом, **время жизни динамического массива**, как и любой динамической переменной – с момента выделения памяти до момента ее освобождения. Область действия элементов массива зависит от места декларации указателя, через который производится работа с его элементами. Область действия и время жизни указателей подчиняются общим правилам для остальных объектов программы.

Динамический массив

Пример

```
#include <malloc.h>
void main()
{
        double *x;
        int n;
                printf("\nВведите размер массива – ");
                scanf("%d", &n);

                if ((x = (double*)calloc(n, sizeof(*x)))==NULL)  {   // Захват памяти
                        puts("Ошибка ");
                        return;
                }
         ...
                // Работа с элементами массива
         ...
        free(x);                                                    // Освобождение памяти
}
```

Динамический массив

***ID* двухмерного массива** – указатель на указатель. В данном случае сначала выделяется память на указатели, расположенные последовательно друг за другом, а затем каждому из них выделяется соответствующий участок памяти под элементы.

Пример

```
. . .
int  **m, n1, n2, i, j;
puts(" Введите размеры массива (строк, столбцов): ");
scanf("%d%d", &n1, &n2);

// Захват памяти для указателей – A (n1=3)
m = (int**)calloc(n1, sizeof(int*));
for (i=0; i<n1; i++)                          // Захват памяти для элементов – B (n2=4)
        *(m+i) = (int*)calloc(n2, sizeof(int));

for ( i=0; i<n1; i++)
        for ( j=0; j<n2; j++)
                m[i] [j] = i+j;               //  *(*(m+i)+j) = i+j;
. . .
for(i=0; i<n; i++) free(m[i]);                // Освобождение памяти
free(m);
. . .
```