Практическая работа 17

Тема: ОДНОМЕРНЫЕ И ДВУМЕРНЫЕ МАССИВЫ

Теоретическая часть

Массивом называется множество элементов одного типа, расположенных в памяти последовательно друг за другом. При первом упоминании о массиве в программе под него сразу выделяется память.

Синтаксис определения массива имеет вид

Тип_элемента имя_массива $[n_1][n_2]...[n_k]$;

где имя массива - идентификатор, определяемый в качестве имени массива, а n_i- размеры массива. Массив называется k-мерным массивом с элементами типа *тип_элемента*. Элементы i-го измерения имеют индексы от 0 до n_i-1. Тип элемента массива может быть одним из основных типов, типом указателя (pointer), типом структуры (struct) или типом объединения (union). Хотя элементы массива не могут быть функциями, они могут быть указателями на функции.

Ниже приведены некоторые примеры определений массива:

```
int page[10]; /* одномерный массив из 10 элементов типа int, пронумерованный с 0 до 9 */
char line[81];/*массив символов или строка, в которую можно записать не более 80 символов */
float big[10][10], sales[10][5][8];
```

Массивы могут быть следующих видов:

1. Локальные. Располагаются в стеке. Например,

```
main(){
int A[10];
//…..
}
```

2. Статические. Располагаются в области данных, глобальных и статических переменных. Например,

```
main(){
static int A[10];
//…..
}
```

3. Глобальные. Располагаются в области данных, глобальных и статических переменных. Например,

```
int A[10];
main(){
//…..
}
```

4. Дальние глобальные. Располагаются в дальней области глобальных переменных. Например,

```
far int A[10];
main(){
//…..
}
```

Двумерные массивы располагаются в памяти по строкам. Начальную строку массива называют нулевой строкой. В общем случае, многомерные массивы располагаются в памяти так, что при последовательном просмотре его элементов последние индексы меняются быстрее.

Например, трехмерный массив intA[3][4][5] располагается в памяти слоями A[0][…][…], …, A[2][…][…].

Каждый слой, как двумерный массив, располагается по строкам. Например, A[0][0][…], …, A[0][3][…].

Массивы могут размещаться только в пределах одного сегмента, то есть общий размер массива в байтах не превышает 64К.

Элементы массива могут стоять в обеих частях операции присваивания.

Задание элемента k-мерного массива реализуется последовательным применением операций индексации:

$x[i_1][i_2]...[i_k]$,

где i_k- целое выражение, при этом $0<=i_k<=n_k-1$, где n_k — размер массива по k-му индексу. Например:

```
page[5]
line[i+j-1]
big[i][j]
```

Операция индексации является левоассоциативной операцией, то есть выполняется в выражении слева направо. Поэтому при обращении к элементу массива вначале выполняется левая операция индексации []. К полученному результату применяется вторая операция индексации [] и т.д.

Инициализация массивов может быть полной или частичной.

Одномерные массивы

1. В случае полной инициализации указывается полный список значений в фигурных скобках.

```
int A[4] = {1, 4, 2, 6};
```

Размеры массивов при полной инициализации можно не указывать. Компилятор сам для себя определит эти размеры и выделит соответствующую память. Найти размер массива можно с помощью операции *sizeof*. Операция возвращает размер всего, что угодно в байтах. В частности, sizeof(A) возвращает размер массива в байтах. Например,

```
int A[] = {1, 4, 2, 6};
int Dim = sizeof(A) / sizeof(int); // 8/2=4
```

Или

```
int Dim = sizeof(A) / sizeof(A[0]); // 8/2=4
```

2. В случае частичной инициализации указывается размер массива и неполный список значений в фигурных скобках. Не инициализированные элементы получают нулевые значения. В случае

```
int A[4] = {1, 4};
```

элементы A[0] и A[1] получили значения, а в A[2] и A[3] записаны нули .

Если список инициализации больше размера массива, то возникнет ошибка компиляции.

```
// int A[4] = {1, 4, 4, 7, 2}; Ошибка
```

Двумерные массивы

1. В случае полной инициализации указывается полный список значений в фигурных скобках. Каждая строка инициализируется в своих фигурных скобках.

```
int A[3][4] ={ {1, 4, 2, 6},
{11, 1 4, 1 2, 1 6},
{1, 4, 2, 6}
};
```

Первый размер массива, то есть количество строк, при полной инициализации можно не указывать.

```
int A[][4] ={ {1, 4, 2, 6},
{11, 1 4, 1 2, 1 6},
{1, 4, 2, 6}
};
```

Компилятор сам определит количество по списку инициализации. Можна найти размер массива или строк с помощью операции *sizeof*. Так sizeof(A) возвращает размер двумерного массива в байтах, а sizeof(A[0]) возвращает размер строки в байтах. Например,

```
int KolStrok = sizeof(A) / sizeof(A[0]); // 24/82=3
```

2. В случае частичной инициализации указываются все размеры массива и неполные списки значений в фигурных скобках.

```
int A[4][4] ={ { 2, 6},
{ 1 4, 1 2, 1 6},
{6}
};
```

Если размер список инициализации больше хотя бы одного размера массива, то возникнет ошибка компиляции.

```
// int A[2][4] = {{1, 4, 4, 7, 2},
{1, 4, 4, 2}}; Ошибка
```

Допускается инициализация двумерного массива одной парой фигурных скобок

```
int A[2][4] = { 1, 4, 4, 7, 2, 1, 4, 4, 2};
```

но такой способ чреват логической ошибкой в случае частичной инициализации.

При определении массива без инициализации все размеры надо указывать явным образом.

Для одномерных массивов типом имени массива является *тип_элемента_массива[]*

Например:

Имя A массива charA[20]; имеет тип char[].

Имя B массива floatB[10] имеет тип float[].

Для двумерных массивов типом имени массива является *тип_элемента_массива[][размер]*. В типе имени массива нет информации о первом размере массива – о количестве строк.

Например:

Имя A массива charA[10][20]; имеет тип char[][20].

Имя B массива charB[100][20] тоже имеет тип char[][20].

Имя C массива charC[20][10] имеет тип char[][10], отличный от типов A и B.

При передаче в функцию одномерного массива в списке фактических аргументов указываются имя массива и размер массива. В имени массива нет информации о размере массива. Компилятор по имени массива может определить только тип элементов массива и адрес начального элемента массива. В списке формальных аргументов указываются только типы. Массивы передаются в функцию по адресу. Это означает, что функция работает с оригиналом массива и может изменять его элементы. Это факт используется для возвращения массивов из функции. С помощью оператора return массив вернуть нельзя.

Примеры прототипов

1. int max(int *A, int Dim);

Можно также писать

int max(int A[], int Dim);

Следующая запись логически не верна, так как размер одномерного массива не входит в тип имени массива.

// int max(int A[100]); ошибка

В списке формальных параметров записи int *A и intA[] равносильны.

2. float scal(float A[], float B[], int Dim);

Пример 1. Функция находит сумму элементов одномерного массива типа int и программу.

```
#include <iostream>

using namespace std;
```

```
int sum( int *A, int Dim); // прототип функции

int main ()
{
	int B[] = {1,2,3,4,5};
	int N = sizeof(B)/sizeof(B[0]);
	cout << " Сумма элементов равна " << sum(B, N) << endl;
	return 0;
}

int sum( int *A, int Dim)
{
int S =0;
for (int i = 0; i < Dim; i++)
S += A[i];
return S;
}
```

Пример 2. Функция находит все четные элементы в одномерном массиве.

```
int VseChot(intA[], intDimA, intChot[], intDimChot);
```

Функция находит четные элементы массива A и помещает их в массив Chot и возвращает количество найденных четных элементов. Если количество четных элементов превысит размер DimChot, то возвращается -1. Массив Chot полностью состоит из четных элементов массива A.

```
int VseChot(int A[], int DimA, int Chot[], int DimChot)
{
	int count = 0;
	for(int i = 0; i < DimA; i++)
		if ( A[i] % 2 == 0) //четное число
			if (count < DimChot)
				Chot[count++] = A[i];
			else
				return -1;
}

int main()
{
	int A[]={1, 2, 4, 6 ,7, 5};
	int B[4];
	int res = VseChot( A, 6, B, 4);
	if(res = = -1)
```

```
        {
        cout << "\n Найдены не все четные элементы массива:";
        for (int i = 0; i < 4; i++)
                cout <<  B[i];
        }
        else
        {
        cout << "\n Перечень четных элементов массива :";
        for (int i = 0; i < 4; i++)
                cout << B[i];
        }
return 0;
}
```

В случае двумерных массивов нужно точно соблюдать совпадение типов фактических и формальных параметров функции.

Например:

1. Функция находит максимальный элемент в массиве

```
int max(int A[][100], int KolStroc, int KolStolb);
```

Данная функция может вызываться только для массивов, у которых второй размер 100. В противном случае, будет ошибка компиляции.

Например, можно вызвать эту функцию для частично инициализированного массива

```
int A[][100] = {{1,3,5}, {15,2,3}};
int res = max( A, 2, 3);
```

2. Функция находит сумму элементов двумерного массива

При передаче двумерного массива здесь использовано явное преобразование типа двумерного массива к типу одномерного массива. Это позволяет вызывать функцию для любых двумерных массивов.

```
int sum(int A[], int KolStroc, int KolStolb)
{
int s= 0;
for (int i = 0; i < KolStroc; i++)
        for (int j = 0; j < KolSolb; j++)
                s += A[i* KolSolb + j];
return s;
};

int main()
```

```
{
	int A[2][3]={{1,4,2}, {4,1,2}};
	int res = sum((int *)A, 2, 3);
	cout << s;
	return 0;
}
```

Практическое задание

1. Напишите функцию, которая находит сумму двух одномерных массивов.
2. Напишите функцию, которая находит произведение одномерного массива на число.
3. Напишите функцию, которая находит минимальное значение в одномерном массиве и возвращает его. Приведите также вариант функции, которая возвращает индекс элемента с минимальным значением.
4. Напишите функцию, которая находит скалярное произведение двух векторов.