2.5. Структурированные типы данных. Понятие массива. Одномерные и двумерные массивы

В математике для удобства записи различных операций часто используют индексированные переменные: векторы, матрицы и т.п. Так, вектор $\vec{c}$ представляется набором чисел (c_1, c_2, ..., c_n), называемых его компонентами, причем каждая компонента имеет свой номер, который принято обозначать в виде индекса. Матрица A – это таблица чисел (a_{ij}, i=1,..., n; j=1,..., m), i – номер строки, j – номер столбца. Операции над матрицами и векторами обычно имеют короткую запись, которая обозначает определенные, порой сложные действия над их индексными компонентами. Например, произведение двух векторов записывается как $\vec{c} \cdot \vec{b} = \sum_{i=1}^{n} c_i b_i$. Произведение матрицы на вектор $\vec{b} = A \cdot \vec{c}, \quad b_i = \sum_{j=1}^{n} a_{ij} \cdot c_j$.

Таким образом, если с группой величин одинакового типа требуется выполнять однообразные действия, им дают одно имя, а различают по порядковому номеру.

Введение индексированных переменных в языках программирования также позволяет значительно облегчить реализацию многих сложных алгоритмов, связанных с обработкой массивов однотипных данных.

Например, использование массивов данных позволяет компактно записывать множество операций с помощью циклов.

В языке Си для этой цели используется сложный тип данных – ***массив***, представляющий собой упорядоченную конечную совокупность элементов одного типа. Число элементов массива называют его *размером*. Каждый элемент массива определяется идентификатором массива и своим порядковым номером – *индексом*. ***Индекс*** – целое число, по которому производится доступ к элементу массива. Индексов может быть несколько. В этом случае массив называют многомерным, а количество индексов одного элемента массива является его размерностью.

Описание массива в программе отличается от описания простой переменной наличием после имени квадратных скобок, в которых задается количество элементов массива. Например, *double a* [10]; – описание массива из 10 вещественных чисел.

При описании массивов квадратные скобки являются элементом синтаксиса, а не указанием на необязательность конструкции.

Размеры массивов предпочтительнее вводить с клавиатуры как значения целочисленных переменных или задавать с помощью именованных констант, поскольку при таком подходе для ее изменения достаточно скорректировать значение константы всего лишь в одном месте программы.

Одномерные массивы

В программе одномерный массив объявляется следующим образом:

тип ID_массива [***размер***] = {список начальных значений};

тип – базовый тип элементов массива (целый, вещественный, символьный); *размер* – количество элементов в массиве.

Список начальных значений используется при необходимости инициализировать данные при объявлении, он может отсутствовать.

При декларации массива можно использовать также атрибуты «класс памяти» и *const*.

Размер массива вместе с типом его элементов определяет объем памяти, необходимый для размещения массива, которое выполняется на этапе компиляции, поэтому размер массива задается только константой или константным выражением. Нельзя задавать массив переменного размера, для этого существует отдельный механизм – динамическое выделение памяти.

Пример объявления массива целого типа: int *a*[5];

Индексы массивов в языке Си начинаются с 0, т.е. в массиве *a* первый элемент: *a*[0], второй – *a*[1], … пятый – *a*[4].

Обращение к элементу массива в программе на языке Си осуществляется в традиционном для многих других языков стиле – записи операции обращения по индексу [] (квадратные скобки), например:

```
a[0]=1;
a[i]++;
a[3]=a[i]+a[i+1];
```

Пример объявления массива целого типа с инициализацией начальных значений:

```
int a[5]={2, 4, 6, 8, 10};
```

Если в группе {…} список значений короче, то оставшимся элементам присваивается 0.

Внимание. В языке Си с целью повышения быстродействия программы отсутствует механизм контроля выхода за границы индексов массивов. При необходимости такой механизм должен быть запрограммирован явно.

Многомерные массивы

Декларация многомерного массива имеет следующий формат:

тип ID[*размер*1][*размер*2]…[*размерN*] =
{ {*список начальных значений*},
{*список начальных значений*},
…
};

Списки начальных значений – атрибут необязательный.

Наиболее быстро изменяется последний индекс элементов массива, поскольку многомерные массивы в языке Си размещаются в памяти компьютера построчно друг за другом (см. следующую тему «Адресная функция»).

Рассмотрим особенности работы с многомерными массивами на конкретном примере двухмерного массива.

Например, пусть приведена следующая декларация двухмерного массива:

int m[3][4];

Идентификатор двухмерного массива – это указатель на массив указателей (переменная типа указатель на указатель: *int* ***m*;).

Поэтому двухмерный массив *m*[3][4]; компилятор рассматривает как массив трех указателей, каждый из которых указывает на начало массива со значениями размером по четыре элемента каждый. В ОП данный массив будет расположен следующим образом:

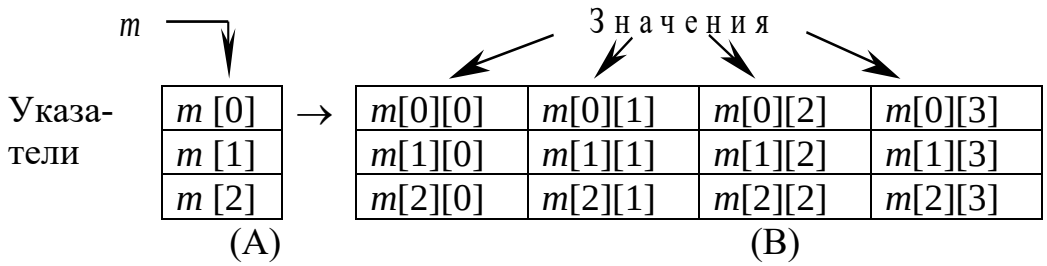

Рис. 10.1. Схема размещения элементов массива *m* размером 3×4

Причем в данном случае указатель *m*[1] будет иметь адрес *m*[0]+4**sizeof*(*int*), т.е. каждый первый элемент следующей строки располагается за последним элементом предыдущей строки.

Приведем пример программы конструирования массива массивов:

```
#include <stdio.h>
void main()
{
        int x0[4] = { 1, 2, 3,4};                  //  Декларация и инициализация
        int x1[4] = {11,12,13,14};                 //  одномерных массивов
        int x2[4] = {21,22,23,24};
        int *m[3] = {x0, x1, x2,};                 // Создание массива указателей
        int i,j;
        for (i=0; i<3; i++) {
                printf("\n Строка %d) ", i+1);
                for (j=0; j<4; j++)
                        printf("%3d", m[ i ] [ j ]);
                }
        }
```

Результаты работы программы:

Строка 1) 1   2   3   4
Строка 2) 11 12 13 14
Строка 3) 21 22 23 24

Такие же результаты будут получены и в следующей программе:

```
#include <stdio.h>
void main()
{
        int i, j;
        int m[3][4] = { { 1, 2, 3, 4}, {11,12,13,14}, {21,22,23,24} };
        for (i=0; i<3; i++) {
                printf("\n %2d)", i+1);
                for (j=0; j<4; j++)
                        printf(" %3d",m[ i ] [ j ]);
        }
}
```

В последней программе массив указателей на соответствующие массивы элементов создается компилятором автоматически, т.е. данные массива располагаются в памяти последовательно по строкам, что является основанием для декларации массива *m* в виде

```
int m[3][4] = {1, 2, 3, 4, 11, 12, 13, 14, 21, 22, 23, 24};
```

Замена скобочного выражения *m*[3][4] на *m*[12] здесь не допускается, так как массив указателей не будет создан.

Таким образом, использование многомерных массивов в языке Си связано с расходами памяти на создание массивов указателей.

Очевидна и схема размещения такого массива в памяти – последовательное (друг за другом) размещение «строк» – одномерных массивов со значениями (векторная организация памяти).

Обращению к элементам массива при помощи операции индексации *m*[*i*][*j*] соответствует эквивалентное выражение, использующее адресную арифметику – *(*(*m*+*i*)+*j*).

Аналогичным образом можно установить соответствие между указателями и массивами с произвольным числом измерений.

Строки как одномерные массивы данных типа char

В языке Си отдельного типа данных «строка символов» нет. Работа со строками реализована путем использования одномерных массивов типа *char*, т.е. строка символов – это одномерный массив символов, заканчивающийся нулевым байтом.

Нулевой байт – это байт, каждый бит которого равен нулю, при этом для нулевого байта определена символьная константа ´\0´ (признак окончания строки, или «*нуль-символ»*). Поэтому если строка должна содержать *k* символов, то в описании массива размер должен быть *k*+1. По положению нуль-символа определяется фактическая длина строки.

Например, char *s*[7]; – означает, что строка может содержать не более шести символов, а последний байт отводится под нуль-символ.

Отсутствие нуль-символа и выход указателя при просмотре строки за ее пределы – распространенная ошибка при работе со строками.

Строку можно инициализировать строковой константой (строковым литералом), которая представляет собой набор символов, заключенных в двойные кавычки. Например:

```
char S[ ] = “Работа со строками”;
```

для данной строки выделено и заполнено 19 байт – 18 на символы и 19-й на нуль-символ.

В конце строковой константы явно указывать символ ´\0´ не нужно. Компилятор добавит его автоматически.

Символ ´\0´ нужно использовать явно тогда, когда символьный массив при декларации инициализируется списком начальных значений, например, следующим образом:

```
char str[10] ={‘V’ , ‘a’, ‘s’, ‘j’ , ‘a’, ‘\0’};
```

или когда строка формируется посимвольно в коде программы. Пример такого формирования приведен в конце этого раздела.

При работе со строками можно пользоваться указателями, например:

```
char *x;
x = "БГУИР";
x = (i>0) ? "положительное" : (i<0) ? "отрицательное" : "нулевое";
```

Такая декларация строки – единственный случай, когда в коде программы можно использовать операцию присваивания явно.

Операция char *str = "БГУИР" создает не строковую переменную, а ***указатель на строковую константу, изменить которую невозможно***, причем это касается не только адреса ОП, но и его размера. Знак равенства перед строковым литералом означает инициализацию, а не присваивание.

Операция присваивания одной строки другой в языке Си не определена (поскольку строка является массивом) и может обрабатываться при помощи оператора цикла (с использованием стандартной библиотечной функций).

Процесс копирования строки *s*1 в строку *s*2 имеет вид

```
char s1[25], s2[25];
for (int i = 0; i <= strlen(s1); i++)
        s2[i] = s1[i];
```

Длина строки определяется с помощью стандартной функции *strlen*, которая вычисляет длину, выполняя поиск нуль-символа (прототип функции приведен ниже). Таким образом, строка фактически просматривается дважды.

А вот следующие действия будут ошибочными:

```
char s1[51];
        s1 = ”Minsk”;
```

Это связано с тем, что *s*1 – константный указатель и не может использоваться в левой части операции присваивания.

Большинство действий со строковыми объектами в Си выполняются при помощи стандартных библиотечных функций, так, для правильного выполнения операции присваивания в последнем примере необходимо использовать стандартную функцию

strcpy(s1, ”Minsk”);

Напомним, что для ввода строк, как и для других объектов программы, обычно используются две стандартные функции:

Функция *scanf* вводит значения для строковых переменных при помощи формата (спецификатора ввода) %*s* до появления первого символа “пробел” (символ «&» перед *ID* строковых данных указывать не надо);

Функция *gets* осуществляет ввод строки, которая может содержать пробелы. Завершается ввод нажатием клавиши *Enter*.

Обе функции автоматически ставят в конец строки нулевой байт.

Вывод строк производится функциями *printf* или *puts* до нулевого байта.

Функция *printf* не переводит курсор после вывода на начало новой строки, а *puts* автоматически переводит курсор после вывода строковой информации в начало новой строки. Например:

```
char Str[30];
printf(“ Введите строку без пробелов : \n”);
scanf(“%s”, Str);
```

или

```
puts(“ Введите строку ”);
gets(Str);
```

Остальные операции над строками, как уже отмечалось ранее, выполняются с использованием стандартных библиотечных функций, декларация прототипов которых находятся в файле ***string.h***.

Приведем наиболее часто используемые стандартные строковые функции.

Функция ***strlen***(*S*) возвращает длину строки (количество символов в строке), при этом завершающий нулевой байт не учитывается, например:

```
char *S1 = ”Минск!\0”, S2[] = ”БГУИР–Ура!”;
printf(“ %d , %d .”, strlen(S1), strlen(S2));
```

Результат выполнения данного участка программы:

6 , 10 .

Функция ***strcpy***(*S*1, *S*2) – копирует содержимое строки *S*2 в строку *S*1.

Функция ***strcat***(*S*1, *S*2) – присоединяет строку *S*2 к строке *S*1 и помещает ее в массив, где находилась строка *S*1, при этом строка *S*2 не изменяется. Нулевой байт, который завершал строку *S*1, заменяется первым символом строки S2.

Функция *int* ***strcmp***(*S*1, *S*2) сравнивает строки *S*1 и *S*2 и возвращает значение <0, если *S*1<*S*2; >0, если *S*1>*S*2; =0, если строки равны, т.е. содержат одно и то же число одинаковых символов.

Функции преобразования строковых объектов в числовые описаны в библиотеке *stdlib.h*. Рассмотрим некоторые из них.

Преобразование строки *S* в число:

– целое: *int* ***atoi***(S);
– длинное целое: *long* ***atol***(S);
– действительное: *double* ***atof***(S);

при возникновении ошибки данные функции возвращают значение 0.

Функции преобразования числа *V* в строку *S*:

– целое: ***itoa***(V, S, kod);
– длинное целое: ***ltoa***(V, S, kod);

$2 \le kod \le 36$, для десятичных чисел со знаком $kod = 10$.

Пример участка кода программы, в котором из строки *s* удаляется символ, значение которого содержится в переменной *c* каждый раз, когда он встречается

```
char s[81], c;
        ...
for( i = j = 0; s[i] != '\0'; i++)
     if( s[i] != c)  s[j++] = s[i];
s[j]='\0';
        ...
```