Лабораторная работа № 6 Обработка динамических массивов

1. Цели лабораторной работы:

- освоить средства моделирования динамических массивов;
- научиться создавать функции для обработки массивов.

2. Последовательность действий

1. Познакомьтесь со стандартными функциями Си, обеспечивающими моделирование динамических массивов (см. ниже раздел 1).
2. Изучите способы моделирования массивов, описанные в разделе 2.
3. Переработайте программу, написанную Вами при выполнении ЛР 5, используя эти способы.
4. Структурируйте Вашу программу. Оставьте в функции main() только действия, связанные с вводом исходных данных и печатью результатов. Решение основной задачи выделите в отдельную функцию. Для организации обмена данными между функциями используйте подходящие способы моделирования массивов (см. разделы 2, 3).
5. Напишите отчёт о работе. Отчёт должен включать:
   - листинги написанных программ;
   - условия и результаты тестирования программ;
   - выводы (изложение полученных Вами в ходе выполнения ЛР новых представлений о возможностях языка С в части обработки массивов).

3. Динамические массивы в Си

3.1. Понятие динамического массива

Динамическим называют массив, размер которого может изменяться в процессе исполнения программы. Объявить динамический массив, используя конструкцию

`<тип> <имя>[<размер>];`

невозможно. Дело в том, что это объявление жёстко привязывает массив к определённому участку памяти. При этом реальный размер массива может быть НЕ БОЛЬШЕ объявленного. Выход за пределы этого участка чреват порчей данных.

Динамический массив не может быть привязан к определённому участку памяти, т.к. невозможно заранее предсказать размер этого участка. Для моделирования динамического массива необходимо иметь возможности

А) резервирования в нужный момент времени блока памяти требуемого размера;
Б) переноса содержимого массива (если оно существует) в новый блок;
В) обеспечения доступа программы к этому блоку;
Г) освобождения ставшего ненужным блока, зарезервированного ранее.

В языке Си возможности А), Б), Г) обеспечиваются стандартными функциями для работы с памятью (заголовочный файл **`<stdlib.h>`**). Описания этих функций приведены ниже (см. Керниган Б., Ритчи Д. Язык программирования Си. Приложение В. В 5. Функции общего назначения: <stdlib. h>).

3.2. Стандартные функции для работы с памятью

`void *calloc(size_t nobj, size_t size)`[1]

Возвращает нетипизированный указатель на место в памяти, отведённое для массива **`nobj`** объектов, каждый из которых имеет размер **`size`** (в байтах), или, если памяти запрашиваемого объёма нет, **`NULL`**. Выделенная область памяти обнуляется.

Пример 1. Использование calloc()

```
1. int *arr;
2. unsigned N;
3. printf("\tВведите размер массива: ");
4. scanf("%u", &N);
5. arr = (int *)calloc( N, sizeof(int) );
6. if ( !arr ) ...Запрошенная память не выделена
7.     else... Память для размещения N объектов типа int[2] выделена и
                очищена (заполнена нулями). arr – адрес первого байта
                выделенного блока.
```

Теперь переменная **`arr`** имеет смысл имени массива, но не является адресной константой. Для доступа к элементам массива **`arr`** можно использовать индексы или сдвиг указателя. Например:

```
int i;
for( i = 0; i < N; i++)
   arr[i] = rand();
```

Или, то же самое:

```
int i, *ptr = arr;
for( i = 0; i < N; i++)
   *ptr++ = rand();
```

Можно и так:

```
int i;
for( i = 0; i < N; i++)
   *arr++ = rand();
arr -= N;
```

ВОПРОС: зачем нужна последняя строка?

`void *malloc(size_t size)`

Возвращает нетипизированный указатель на место в памяти для объекта размера **`size`** байтов или, если памяти запрашиваемого объёма нет, NULL. Выделенная область памяти не инициализируется.

---

[1] Тип **`size_t`** есть тип беззнакового целого, возвращаемого оператором **`sizeof`**.

[2] Вовсе не обязательно использовать этот блок для размещения элементов заявленного типа.

Пример 2. Использование malloc()

```
1. int *arr;
2. unsigned N;
3. printf("\tВведите размер массива: ");
4. scanf("%u", &N);
5. arr = (int*)malloc( N * sizeof(int) );
6. if ( !arr ) ...Запрошенная память не выделена
7.     else ...Выделен блок памяти размера N*sizeof(int) байтов.
                Память не очищена.
```

Перечисленные выше варианты манипулирования именем **arr** допустимы и в этом случае.

void *realloc(void *p, size_t size)

Заменяет на **size** размер объекта, на который указывает **p**. Для части, размер которой равен наименьшему из старого и нового размеров, содержимое не изменяется. Если новый размер больше старого, дополнительное пространство не инициализируется, **realloc** возвращает нетипизированный указатель на новый блок памяти заявленного размера. Если требования не могут быть удовлетворены, возвращает **NULL.** Если **p == NULL**, то работает как **malloc**.

Пример 3. Использование realloc()

```
int *arr;
unsigned N;
printf("\tВведите размер массива: ");
scanf("%u", &N);
arr = (int*)malloc( N * sizeof(int) );
.........................Работа с массивом arr.
```

Потребовалось увеличить размер массива на **M** элементов:

```
int *ptr = (int*)realloc( arr, (N + M) * sizeof(int) );
if ( !ptr ) ...Запрошенная память не выделена
    else          Выделен запрошенный блок памяти.
       { arr = ptr;
.........................Работа с расширенным arr
       }
```

ВОПРОС: зачем нужна переменная ptr?

void free(void *p)

Значения не возвращает. Освобождает область памяти, на которую указывает **p**; эта функция ничего не делает, если **p** равно **NULL**. Параметр **p** должен быть указателем на область памяти, ранее выделенную одной из функций: **calloc, malloc** или **realloc**.

Простой пример обработки динамического массива см. в папке **Пример**.

3.3. Способы моделирования массивов

СОВЕТ. Изучая этот раздел, пытайтесь фиксировать на бумаге в виде рисунков свои представления о размещении элементов массивов в памяти.

Уже понятно, что модель одномерного массива (динамического/статического) – структурированный блок памяти. Имя массива – указатель на первый байт блока. Доступ к элементам массива обеспечивается сдвигом этого указателя. Величина сдвига определяется индексом и размером типа элемента массива. Понятия многомерного массива как структурного типа данных в Си нет. Так, двумерный массив моделируется одномерным массивом элементов, являющихся, в свою очередь, одномерными массивами чисел.

Объявление **`int Arr[m][n]`** с «точки зрения» компилятора есть объявление массива из **`m`** элементов, каждый из которых является блоком (строкой) из **`n`** чисел типа **`int`**. Выражение **`Arr[i]`** возвратит *адрес* начала **`i`**-того блока (строки), т.е. значение адреса, сдвинутого относительно **`Arr`** на **`i*n*sizeof(int)`** байтов. Таким образом, выражение **`Arr[i]`** эквивалентно выражению ***(`Arr+i`)**, где **`i`** – число блоков (индекс строки). Величина сдвига адреса в байтах определяется длиной элемента массива **`Arr`**, т.е. блока из **`n`** чисел типа **`int`**. Чтобы получить доступ к **`j`**-тому элементу этого блока (числу типа **`int`**), нужно сдвинуть этот указатель ещё на **`j*sizeof(int)`** байтов: ***(`Arr+i)+j`**. Теперь величина сдвига адреса в байтах определяется длиной типа элемента строки (в нашем примере **`int`**). Наконец, чтобы получить значение **`j`**–того элемента **`i`**-той строки, нужно выполнить операцию разыменования. Т.е. выражение **`Arr[i][j]`** эквивалентно выражению ***(*(`Arr+i)+j`)**.

Обратите внимание! Объявленный так двумерный массив занимает в оперативной памяти монолитный блок размера **`m*n*sizeof(int)`** байтов. С ним можно работать как с одномерным массивом из **`m*n`** элементов. И наоборот, имеющийся одномерный массив из **`m*n`** элементов можно обрабатывать как двумерный массив из **`m`** строк и **`n`** столбцов.

Однако, используя аппарат указателей и функции для работы с памятью, можно моделировать многомерный массив так, что его фрагменты в общем случае могут быть разбросаны (физически) по разным “углам” оперативной памяти, но (логически) восприниматься и обрабатываться программой как единый объект. Вот пример такой модели.

```
int *Arr[m]; // Объявлен массив указателей на int.
int i;
for( i = 0; i < m; i++ )
  Arr[i] = (int*)calloc( n, sizeof(int) );
/*  Массив Arr заполнен значениями указателей на блоки памяти размером в n элементов типа int. */
```

Эти блоки в общем случае не примыкают друг к другу. Тем не менее, с именем **`Arr`** можно обращаться как с именем двумерного массива. Выражение **`Arr[i][j]`** возвратит значение **`j`**–того элемента **`i`**-той строки матрицы.

В приведённом примере **`Arr`** — статический массив указателей на строки моделируемого двумерного массива. Размеры этих строк *могут изменяться* в процессе исполнения про-

граммы. В общем случае длины строк могут быть *различными*. Т.е., созданный так массив строк есть нечто более сложное, чем матрица.

Можно создавать ещё более “свободные” динамические модели двумерных (и не только) массивов.

3.4. Массив как параметр функции

Пусть нужно написать функцию для обработки числового массива.

Если это одномерный массив, то прототип функции может быть таким:

```
<тип> <имяФ>( <тип> <имяМ>[], int <размер>,… );
```

Или, что то же самое:

```
<тип> <имяФ>( <тип> *<имяМ>, int <размер>,… );
```

Такая функция может обрабатывать как статические, так и динамические массивы.

Прототип функции, обрабатывающей двумерный массив, зависит от способа моделирования массива.

1. Пусть требуется обрабатывать статический двумерный массив. Тогда прототип функции должен иметь вид:

```
<тип> <имяФ>(int <разм1>, int <разм2>, <тип> <имяМ>[][<разм2>],… );
```

<разм2> должен быть количеством столбцов.

ПОРЯДОК ПАРАМЕТРОВ ИМЕЕТ ЗНАЧЕНИЕ. Размеры массива должны предшествовать объявлению его имени.

2. Если в вызывающей программе двумерный массив моделируется с использованием массива указателей на строки, то прототип функции может иметь вид:

```
<тип> <имяФ>(<тип> **<имяМ>, int <разм1>, int <разм2>, … );
```

ИЛИ

```
<тип> <имяФ>(<тип> *<имяМ>[], int <разм1>, int <разм2>, … );
```

ПОРЯДОК ПАРАМЕТРОВ НЕ ИМЕЕТ ЗНАЧЕНИЯ.