# Управление памятью

##  Указатели

Рассмотрим такую задачу: в файле записаны целые числа. Надо отсортировать их и запи- сать в другой файл. Проблема заключается в том, что заранее неизвестно, сколько в файле та- ких чисел. В такой ситуации есть два варианта:

1. Выделить память с запасом, если известно, например, что этих чисел гарантированно не более 1000.
2. Использовать динамическое выделение памяти – сначала определить, сколько чисел в массиве, а потом выделить ровно столько памяти, сколько нужно.

Наиболее грамотным решением является второй вариант (использование динамических масси- вов). Для этого надо сначала поближе познакомиться с указателями.

**Указатель** – это переменная, в которой хранится адрес другой переменной или участка па- мяти.

Указатели являются одним из основных понятий языка Си. В такие переменные можно записы- вать адреса любых участков памяти, на чаще всего – адрес начального элемента динамического массива. Что же можно делать с указателями?

|  |  |  |  |
| --- | --- | --- | --- |
| **Объявить** | **char \*pC; int \*pI; float \*pF;** | Указатели объявляются в списке переменных, но перед их именем ставится знак **\**.*** Указатель всегда указывает на переменную того типа, для которого он был объявлен. Например, **pC** может указывать на любой символ, **pI** – на любое целое число, а **pF** – на  любое вещественное число. | |
| **Присвоить адрес** | **int i, \*pI;**  **...**  **pI = &i;** | Такая запись означает: «записать в указатель **pI**  адрес переменной **i**». Запись **&i** обозначает адрес  переменной **i*.*** | |
| **Получить значение по этому адресу** | **float f, \*pF;**  **...**  **f = \*pF;** | Такая запись означает: «записать в переменную **f** то вещественное число, на которое указывает указа- тель **pF**». Запись **\*pF** обозначает содержимое ячей-  ки, на которую указывает **pF*.*** | |
| **Сдвинуть** | **int \*pI;**  **...**  **pI ++; pI += 4;**  **-- pI; pI -= 4;** | В результате этих операций значение указателя ме- няется особым способом: **pI++** сдвигает указатель на РАЗМЕР ЦЕЛОГО ЧИСЛА, то есть на 4 байта, а не на 1 как можно подумать. А запись **pI+=4** или **pI=pI+4** сдвигает указатель на 4 целых числа  дальше, то есть на 16 байт. | |
| **Обнулить** | **char \*pC;**  **...**  **pC = NULL;** | Если указатель равен нулевому адресу **NULL**, это обычно означает, что указатель недействительный.  По нему нельзя ничего записывать (это вызывает сбой программы компьютера). | |
| **Вывести на экран** | **int i, \*pI;**  **...**  **pI = &i;**  **printf("Адр.i =%p", pI);** | | Для вывода указателей использу- ется формат **%p*.*** |

Итак, что надо знать про указатели:

* + указатель – это переменная, в которой записан адрес другой переменной;
  + при объявлении указателя надо указать тип переменных, на которых он будет указывать, а перед именем поставить знак **\***;
  + знак **&** перед именем переменной обозначает ее адрес;
  + знак **\*** перед указателем в рабочей части программы (не в объявлении) обозначает значе- ние ячейки, на которую указывает указатель;
  + нельзя записывать по указателю, который указывает непонятно куда – это вызывает сбой программы, поскольку что-то стирается в памяти;
  + для обозначения недействительного указателя используется константа **NULL**;
  + при изменении значения указателя на **n** он в самом деле сдвигается к **n*-***ому следующему числу данного типа, то есть для указателей на целые числа на **n\*sizeof(int)** байт;
  + указатель печатаются по формату **%p**.

Теперь вам должно быть понятно, что многие функции ввода типа **scanf** и **fscanf** в самом деле принимают в параметрах адреса переменных, например

|  |
| --- |
| **scanf ( "%d", &i);** |

##  Динамическое выделение памяти

Динамическими называются массивы, размер которых неизвестен на этапе написания программы. Прием, о котором мы будем говорить, относится уже не к стандартному языку Си, а к его расширению Си ++. Существуют и стандартные способы выделения памяти в языке Си (с помощью функций **malloc** и **calloc**), но они не очень удобны.

Следующая простейшая программа, которая использует динамический массив, вводит с клавиатуры размер массива, все его элементы, а затем сортирует их и выводит на экран.

|  |
| --- |
| **#include <stdio.h> main()**  **{** |
| **int N; // размер массива (заранее неизвестен) int \*A; // указатель для выделения памяти** |
| **printf ("Размер массива > "); // ввод размера массива**  **scanf ("%d", &N);** |
| **A = new int [N]; // выделение памяти**  **if ( A == NULL ) { // если не удалось выделить**  **printf("Не удалось выделить память");**  **return 1; // выход по ошибке, код ошибки 1**  **}** |
| **for (i = 0; i < N; i ++ ) { // дальше так же, как для массива**  **printf ("\nA[%d] > ", i+1);**  **scanf ("%d", &A[i]);**  **}**  **// здесь сортировка и вывод на экран** |
| **delete A; // освободить память** |
| **}** |

Итак, мы хотим выделить в памяти место под новый массив целых чисел во время работы программы. Мы уже знаем его размер, пусть он хранится в переменной **N** (это число обязатель- но должно быть больше нуля)***.*** Оператор выделения памяти **new** вернет нам адрес нового выде- ленного блока, и для работы с массивом нам надо где-то его запомнить. Вы уже знаете, что есть специальный класс переменных для записи в них адресов памяти – указатели.

###  Что новенького?

* + динамические массивы используются тогда, когда на момент написания программы раз- мер массива неизвестен
  + для того, чтобы работать с динамическим массивом, надо объявить указатель соответст- вующего типа (в нем будет храниться адрес первого элемента массива);

**int \*A;**

* + когда требуемый размер массива стал известен, надо использовать оператор **new** языка Си++, указав в скобках размер массива (в программе для краткости нет проверки на по- ложительность **N**) ;

**A = new int[N];**

* + **нельзя** использовать оператор **new** при отрицательном или нулевом **N**;
  + после выделения памяти надо проверить, успешно ли оно прошло; если память выделить не удалось, то значение указателя будет равно **NULL**, использование такого массива при- ведет к сбой программы;
  + работа с динамическим массивом, на который указывает указатель **А**, идет также, как и с обычным массивом размера **N** (помните, что язык Си не следит за выходом за границы массива);
  + после использования массива надо освободить выделенную память, вызвав оператор

**delete A;**

* + после освобождения памяти значение указателя не изменяется, но использовать его уже нельзя, потому что память освобождена;
  + учитывая, что при добавлении числа к указателю он сдвигается на заданное число ячеек ЗАДАННОГО ТИПА, то следующие записи равносильны и вычисляют адрес **i*-***ого эле- мента массива:

**&A[i] и A+i**

Отсюда следует, что **A** совпадает с **&A[0]**, и поэтому имя массива можно использовать как адрес его начального элемента;

###  Ошибки, связанные с выделением памяти

Самые тяжелые и трудно вылавливаемые ошибки в программах на языке Си связаны именно с неверным использованием динамических массивов. В таблице перечислены наиболее тяжелые случаи и способы борьбы с ними.

|  |  |
| --- | --- |
| Ошибка | Причина и способ лечения |
| **Запись в чужую об- ласть памяти** | Память была выделена неудачно, а массив используется.  **Вывод:** надо всегда делать проверку указателя на **NULL*.*** |
| **Повторное удаление** | Массив уже удален и теперь удаляется снова. |

|  |  |
| --- | --- |
| **указателя** | **Вывод:** если массив удален из памяти, обнулите указатель –  ошибка быстрее выявится. |
| **Выход за границы массива** | Запись в массив в элемент с отрицательным индексом или ин- дексом, выходящим за границу массива |
| **Утечка памяти** | Неиспользуемая память не освобождается. Если это происходит в функции, которая вызывается много раз, то ресурсы памяти скоро будут израсходованы.  **Вывод**: убирайте за собой «мусор» (освобождайте память). |

##  Выделение памяти для матрицы

Для выделения памяти под одномерный массив целых чисел нам потребовался указатель на целые числа. Для матрицы надо выделить указатель на массив целых чисел, который объяв- ляется как

|  |
| --- |
| **int \*\*A;** |

Но лучше всего сразу объявить новый тип данных - указатель на целое число. Новые типы объ- являются директивой **typedef** вне всех процедур и функций (там же, где и глобальные пере- менные).

|  |
| --- |
| **typedef int \*pInt;** |

Этой строкой мы сказали компилятору, что любая переменная нового типа **pInt** представляет собой указатель на целое число или адрес массива целых чисел. К сожалению, место для мат- рицы не удается так же просто выделить в памяти, как мы делали это для одномерного массива.

Если написать просто

|  |
| --- |
| **int M = 5, N = 7;**  **pInt \*A;**  **A = new int[M][N]; // ошибочная строка** |

компилятор выдает множество ошибок. Связано это с тем, что ему требуется заранее знать дли- ну одной строки, чтобы правильно расшифровать запись типа **A[i][j]**. Ниже рассмотрены три способа решения этой проблемы.

###  Известный размер строки

Если размер строки матрицы известен, а неизвестно только количество строк, можно по- ступить так: ввести новый тип данных – строка матрицы. Когда количество строк станет из- вестно, с помощью оператора **new** выделяем массив таких данных.

|  |
| --- |
|  |
| **typedef int row10[10]; // новы тип: массив из 10 элементов** |
| **main()**  **{**  **int N;** |
| **row10 \*A; // указатель на массив (матрица)** |
| **printf ( "Введите число строк "); scanf ( "%d", &N );** |
| **A = new row10[N]; // выделить память на N строк** |
|  |

|  |
| --- |
| **A[0][1] = 25; // используем матрицу, как обычно**  **printf("%d", A[2][3]);** |
| **delete A; // освобождаем память** |
| **}** |

###  Неизвестный размер строки

Пусть размеры матрицы **M** и **N** заранее неизвестны и определяются в ходе работы про- граммы. Тогда можно предложить следующий способ выделения памяти под новую матрицу. Поскольку матрицу можно рассматривать как массив из строк-массивов, объявим **M** указателей и выделим на каждый из них область памяти для одномерного массива размером **N** (то есть, на одну строку). Сами эти указатели тоже надо представить в виде динамического массива. Опре- делив требуемые размеры матрицы, мы выделяем сначала динамический массив указателей, а потом на каждый указатель – место для одной строки.

|  |
| --- |
| **typedef int \*pInt; // новый тип данных: указатель на целое** |
| **main()**  **{**  **int M, N, i;** |
| **pInt \*A; // указатель на указатель** |
| **// ввод M и N**  **A = new pInt[M]; // выделить память под массив указателей** |
| **for ( i = 0; i < M; i ++ ) // цикл по всем указателям**  **A[i] = new int[N]; // выделяем память на строку i** |
| **// работаем с матрицей A, как обычно** |
| **for ( i = 0; i < M; i ++ ) // освобождаем память для всех строк**  **delete A[i];**  **delete A; // освобождаем массив указателей** |
| **}** |

В рассмотренном выше случае на каждую строку выделяется свой участок памяти. Можно по- ступить иначе: сначала выделим область памяти сразу на всю матрицы и запишем ее адрес в **A[0]**. Затем расставим указатели так, чтобы **A[1]** указывал на **N+1*-***ый элемент с начала блока (начало строки 1), **A[2]** – на **2N+1*-***ый (начало строки 2) и т.д. Таким образом, в памяти выде- ляется всего два блока – массив указателей и сама матрица.

|  |
| --- |
| **typedef int \*pInt; main()**  **{**  **int M, N, i;** |
| **pInt \*A; // указатель на указатель** |
| **// ввод M и N**  **A = new pInt[M]; // память на массив указателей** |
| **A[0] = new int [M\*N]; // память для матрицы**  **for ( i = 1; i < M; i ++ ) // расставляем указатели**  **A[i] = A[i-1] + N;** |
| **// работаем с матрицей**  **delete A[0]; // освобождаем матрицу** |
| **delete A; // освобождаем указатели** |
| **}** |