**Практическое занятие № 4**

Целью работы является изучение указателей, возможностей динамического выделения и освобождения памяти. Функции **malloc()** , **realloc()**, **free().** Операторы **new** и **delete**.

**Задание**

Написать программу, реализующую следующий алгоритм. Имеется указатель на массив из одного элемента. Пользователь вводит число. Если оно не равно нулю, число записывается в массив. Далее пользователь вводит второе число, если оно не 0, размер массива увеличивается в два раза до двух элементов и в массив записывается второе число. Затем пользователь вводит третье число, если оно не 0, размер массива увеличивается в два раза до четырех элементов и в массив и записываются третье число. И так далее, пока пользователь не введет ноль. Если размер массива достаточен для записи очередного числа, то память не выделяется.

Ввод чисел организовать построчно по одному в каждой строке. В случаях выделения памяти сразу после введенного числа вывести новый размер массива в той же строке.

Решить задачу с помощью операторов new и delete. Обеспечить адекватную реакцию программы в случае ошибки выделения памяти.

**Общие сведения**

**Указатель** — это переменная, предназначенная для хранения адреса. Указатель инициализируется не значением одного из множества типов данных в C++, а адресом, например адресом другой переменной, которая была объявлена в коде ранее. Пусть имеется следующая переменная:

int iVal = 7;

Для инициализации указателя на эту переменную нужен лишь ее адрес — адрес первого байта, с которого начинается выделенная под нее память. Указатель на эту переменную можно объявить и инициализировать так:

int\* iPtr = &iVal;

Здесь int - тип данных указателя, \* - звездочка, определяющая тип «указатель», & - операция получения адреса. Используя ее, мы извлекаем адрес переменной iVal и присваиваем ее указателю. Обратите внимание, что тип, используемый при объявлении указателя, в точности должен соответствовать типу переменной, адрес которой мы присваиваем указателю.

Теперь, когда мы имеем указатель на переменную *iVal* мы можем оперировать ее значением не только непосредственно с помощью самой переменной, но и с помощью указателя на нее. Посмотрим, как это работает на простом примере:

// выведем на экран значение переменной iVal

cout << iVal; // вариант без указателя

cout << \*iPtr; // вариант с указателем

Используя указатель можно оперировать со значением переменной, на которую он указывает. Для этого используется операция разыменование указателя \*:

(\*iPtr)++; // то же самое, что и iVal++

Рассмотрим одномерный массив заданной длины и инициализируем его элементы:

void main() {

const int size = 7;

int iArray[size]; // объявление массива

for (int i = 0; i != size; i++)

iArray[i] = i; // инициализация элементов массива

}

А теперь будем обращаться к элементам массива, используя указатели:

int\* arrPtr = iArray;

for (int i = 0; i != size; i++) {

std::cout << \*(arrPtr + i) << "\n";

Здесь мы инициализируем указатель arrPtr адресом начала массива iArray. Надо помнить, то использование идентификатора массива без указания квадратных скобок эквивалентно указанию адреса его первого элемента, то есть запись iArray эквивалентно &iArray[0]. Затем в цикле мы выводим элементы, обращаясь к каждому с помощью начального адреса и смещения. То есть:

\*(arrPtr + 0) — это тот же самый нулевой элемент, смещение нулевое (i = 0),

\*(arrPtr + 1) – первый элемент (i = 1), и так далее.

Запись array[i] эквивалентна записи \*(array + i). Никто не запрещает использовать их комбинированно: (array + i)[1] — в этом случае смещение идет на i, и еще на единичку. Однако, в данном случае перед выражением (array + i) ставить **\*** не нужно. Наличие скобок это компенсирует.

***Операторы malloc()*** *,* ***realloc()****,* ***free()***

Очень часто условия задач не позволяют программисту заранее определить объем оперативной памяти, требуемой для хранения данных, используемых в программе. Конечно, можно зарезервировать память «с запасом», рассчитывая на максимально возможный объем данных, однако более эффективным является использование динамического выделения памяти в процессе работы программы, когда становится известным требуемый для размещения данных размер участка памяти. В этом методе память для хранения информации выделяется из свободной области памяти по мере надобности и освобождается, когда надобность в ней исчезла. Область свободной памяти называется кучей (heap) и используется для запросов на динамическое выделение памяти. Преимуществом использования динамической памяти служит то, что одна и та же память после ее освобождения может быть использована для хранения другой информации в процессе исполнения программы.

В языке C++ в библиотеке **сstdlib** имеются унаследованные от языка C функции динамического выделения памяти **malloc** (memory allocation) и **realloc**, которые находят в оперативной памяти непрерывный участок требуемой длины и возвращают начальный адрес этого участка. Функции динамического распределения памяти:

**void\* malloc(размер-памяти-в-байтах)**

**void\* realloc(void \*p, размер-памяти-в-байтах)**

Функция **malloc**() возвращает адрес на первый байт свободной области памяти заданного размера.

Функция **realloc**() изменяет величину ранее выделенной памяти, на которую ссылается указатель p, на новую величину указанного размера. Размер памяти задается в байтах и может быть больше или меньше оригинала. При возрастании размера требуемой памяти старое содержимое копируется в новую область, и информация не теряется.

Если памяти недостаточно, то результатом вызова функций **malloc**() и **realloc**() будет нулевой указатель (**NULL**). В более поздних версиях C++ в качестве нулевого указателя вместо **NULL** рекомендуется использовать константу **nullptr**.

Результат вызова функции, как правило, присваивается указателю, тип которого соответствует типу данных. Например, для выделения памяти под массив целых чисел надо описать [указатель](http://prog-cpp.ru/c-pointers/" \t "_blank" \o "Указатели в языке Си), представляющий собой начальный адрес хранения элементов массива:

int \*p;  // указатель на тип int

Поскольку функции в качестве возвращаемого значения имеют указатель на пустой тип void, требуется явное приведение типа возвращаемого значения. Для этого имя типа возвращаемого значения указывается в круглых скобках перед именем переменной, выражения или функции.

Для определения размера массива в байтах, используемого в качестве аргумента функции **malloc**() требуется количество элементов умножить на размер одного элемента. Для точного определения размера элемента в общем случае рекомендуется использование оператора **sizeof**, который определяет количество байт, занимаемое объектом. При этом можно указать либо сам объект, либо его тип в скобках, например:

**sizeof(тип);**

**sizeof объект; или sizeof ( объект );**

Применение sizeof к массиву дает количество байтов, занимаемых массивом, а не количество его элементов и не размер в байтах каждого из них.

Пример выделения памяти под массив целых чисел, содержащих n элементов:

p = (int\*) malloc(n\*sizeof(int));

Здесь (int\*) – приведение типов; n – количество элементов массива.

Память, динамически выделенная с использованием функций **malloc**(), **realloc**(), может быть освобождена с использованием функции

**free**(указатель);

Функция **free**() освобождает память, но при этом не изменяет значение указателя. Правилом хорошего тона считается освобождение динамически выделенной памяти, если в дальнейшем она не используется. Если динамически выделенная память не освобождается явным образом функцией **free**(), она будет освобождена по завершении выполнения программы.

Пример: Динамическое выделение памяти для одномерного массива целых чисел.

#define \_CRT\_SECURE\_NO\_WARNINGS //для совместимости с классическими функциями

#include <stdio.h> // функции стандартного ввода-вывода

#include <conio.h> // для \_getch

#include <cstdlib> //для malloc

int main() {

int\* p; // указатель на массив

int i, n;

printf("Enter array size: ");

scanf\_s("%d", &n);

p = (int\*)malloc(n \* sizeof(int)); // Выделение памяти

if (p == NULL) {

printf("Allocation error.");

return (1);

}

// Ввод элементов массива

for (i = 0; i < n; i++) {

printf\_s("Array[%d] = ", i);

scanf("%d", &p[i]);

}

// Вывод элементов массива

for (i = 0; i < n; i++)

printf("%d ", p[i]);

free(p);

\_getch(); // Ожидание нажатия клавиши

}

При динамическом выделении памяти для двумерных массивов с помощью функции **malloc**() компилятору явно не указывается количество элементов в строке и столбце, поэтому традиционное обращение к элементу путем указания индекса строки и индекса столбца p[i][j] будет некорректным.

Правильное обращение к элементу с использованием указателя будет выглядеть так:

\*(p+i\*m+j),

где p - указатель на массив, m - количество столбцов, i - номер строки, j - номер столбца.

Пример ввода элемента массива:

scanf("%d", (p+i\*m+j)); // указан адрес элемента

Пример вывода элемента массива:

printf("%d ", \*(p+i\*m+j)); // указан сам элемент.

Возможен также способ динамического выделения памяти под двумерный массив с переменным размером строк - с использованием массива указателей. Для этого необходимо:

* описать или выделить блок оперативной памяти под массив указателей;
* выделить блоки оперативной памяти под одномерные массивы, представляющие собой строки искомой матрицы (могут быть разной длины);
* записать адреса строк в массив указателей.

Фактически такой свободный массив представляет собой одномерный массив указателей на одномерные массивы данных.

Для размещения в оперативной памяти матрицы со строками разной длины может потребоваться дополнительный массив, в котором будут храниться размеры строк.

***Операторы new и delete***

В С++ добавлены другие операторы работы с динамической памятью: **new** и **delete**:

**new**  — для выделения памяти;

**delete** — для освобождения памяти.

Оператор new используется в следующих формах:

**new тип;** — для переменных

**new тип[размер];** — для массивов

Память может быть выделена для одного объекта или для массива любого типа, в том числе типа, определенного пользователем. Результатом выполнения операции **new** будет указатель на отведенную память, или исключение **std::bad\_alloc** в случае ошибки. Можно отменить вызов исключения при ошибке выделения памяти, обеспечив возврат нулевого указателя (**nullptr**). Для этого надо использовать специальную форму оператора: **new (nothrow)**.

Пример:

int \*iPtr;

double \* dPtr;

iPtr = new int; //выделение памяти для одного числа

dPtr = new (nothrow) double[10]; //выделение памяти для массива из 10 чисел

if ( dPtr == nullptr ) {

cout << "Недостаточно памяти" << endl;

Оператор **new** автоматически вычисляет размер необходимой памяти. Нет необходимости в использовании оператора **sizeof()**.

Если объект является массивом, возвращается указатель на начальный элемент массива. Например, обе операции **new int** и **new int[1]** возвратят int\*, Чтобы создать динамический двумерный массив, нам надо создать общий динамический массив указателей, а затем его элементы - вложенные динамические массивы.  Способ выделения памяти под двумерный массив, когда обе его размерности задаются на этапе выполнения программы, приведен ниже:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

int main() {

int nstr, nstb;

std::setlocale(LC\_ALL, "rus");

std::cout << " Количество строк = "; std::cin >> nstr; //ввод кол. строк

std::cout << " Количество столбцов = "; std::cin >> nstb; //ввод кол. столбцов

int\*\* a = new int\* [nstr]; // выделение памяти для массива указателей на строки

for (int i = 0; i < nstr; i++) //цикл по строкам

a[i] = new int[nstb]; //выделение памяти под i-ю строку, состоящую из nstr чисел

for (int i = 0; i < nstr; i++) { //ввод массива

for (int j = 0; j < nstb; j++) {

std::cout << "\na(" << i << "," << j << ")="; std::cin >> a[i][j];

}

}

std::cout << "\nМассив a\n"; //вывод массива

for (int i = 0; i < nstr; i++) {

for (int j = 0; j < nstb; j++) {

std::cout.width(5); std::cout << a[i][j];

}

std::cout << "\n";

}

delete[] a; //освобождение памяти

}

Освобождение памяти связано с тем, как выделялась память – для одного элемента или для нескольких. В соответствии с этим существуют две формы применения **delete**:

**delete указатель;** — для одного элемента

**delete[] указатель;** — для массивов.

Для приведенного выше случая освободить память необходимо следующим образом:

delete iPtr; или delete[] dPtr;