ПРАКТИЧЕСКАЯ РАБОТА №4
ДИНАМИЧЕСКОЕ ВЫДЕЛЕНИЕ ПАМЯТИ ДЛЯ ОБЪЕКТОВ В ЯЗЫКЕ С++

ЦЕЛЬ ПРАКТИЧЕСКОЙРАБОТЫ:

Целью данной практической работы является изучение динамического выделения и удаления памяти, а также работа с указателями в языке С++.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:

Прежде чем углубиться в объектно-ориентированную разработку, нам придется сделать небольшое отступление о работе с памятью в программе на С++. Мы не сможем написать сколько-нибудь сложную программу, не умея выделять память во время выполнения и обращаться к ней.

В С++ объекты могут быть размещены либо статически – во время компиляции, либо динамически – во время выполнения программы, путем вызова функций из стандартной библиотеки. Основная разница в использовании этих методов – в их эффективности и гибкости. Статическое размещение более эффективно, так как выделение памяти происходит до выполнения программы, однако оно гораздо менее гибко, потому что мы должны заранее знать тип и размер размещаемого объекта. К примеру, совсем не просто разместить содержимое некоторого текстового файла в статическом массиве строк: нам нужно заранее знать его размер. Задачи, в которых нужно хранить и обрабатывать заранее неизвестное число элементов, обычно требуют динамического выделения памяти.

До сих пор во всех наших примерах использовалось статическое выделение памяти. Скажем, определение переменной val

```
int ival = 1024;
```

заставляет компилятор выделить в памяти область, достаточную для хранения переменной типа int, связать с этой областью имя ival и поместить туда значение 1024. Все это делается на этапе компиляции, до выполнения программы.

С объектом ival ассоциируются две величины: собственно значение переменной, 1024 в данном случае, и адрес той области памяти, где хранится это значение. Мы можем обращаться к любой из этих двух величин. Когда мы пишем:

```
int ival2 = ival + 1;
```

то обращаемся к значению, содержащемуся в переменной ival: прибавляем к нему 1 и инициализируем переменную ival2 этим новым значением, 1025. Каким же образом обратиться к адресу, по которому размещена переменная?

C++ имеет встроенный тип “указатель”, который используется для хранения адресов объектов. Чтобы объявить указатель, содержащий адрес переменной ival, мы должны написать:

```
int *pint; // указатель на объект типа int
```

Существует также специальная операция взятия адреса, обозначаемая символом &. Ее результатом является адрес объекта. Следующий оператор присваивает указателю pint адрес переменной ival:

```
int *pint;
pint = &ival; // pint получает значение адреса ival
```

Мы можем обратиться к тому объекту, адрес которого содержит pint (ival в нашем случае), используя операцию разыменования, называемую также косвенной адресацией. Эта операция обозначается символом *. Вот как можно косвенно прибавить единицу к ival, используя ее адрес:

```
*pint = *pint + 1; // неявно увеличивает ival
```

Это выражение производит в точности те же действия, что и

```
ival = ival + 1; // явно увеличивает ival
```

В этом примере нет никакого реального смысла: использование указателя для косвенной манипуляции переменной ival менее эффективно и менее наглядно. Мы привели этот пример только для того, чтобы дать самое начальное представление об указателях. В реальности указатели используют чаще всего для манипуляций с динамически размещенными объектами.

Основные отличия между статическим и динамическим выделением памяти таковы:

- статические объекты обозначаются именованными переменными, и действия над этими объектами производятся напрямую, с использованием их имен. Динамические объекты не имеют собственных имен, и действия над ними производятся косвенно, с помощью указателей;
- выделение и освобождение памяти под статические объекты производится компилятором автоматически. Программисту не нужно самому заботиться об этом. Выделение и освобождение памяти под динамические объекты целиком и полностью возлагается на программиста. Это достаточно сложная задача, при решении которой легко наделать ошибок. Для манипуляции динамически выделяемой памятью служат операторы new и delete.

Оператор new имеет две формы. Первая форма выделяет память под единичный объект определенного типа:

```
int *pint = new int(1024);
```

Здесь оператор new выделяет память под безымянный объект типа int, инициализирует его значением 1024 и возвращает адрес созданного объекта. Этот адрес используется для инициализации указателя pint. Все действия над таким безымянным объектом производятся путем разыменовывания данного указателя, т.к. явно манипулировать динамическим объектом невозможно.

Вторая форма оператора new выделяет память под массив заданного размера, состоящий из элементов определенного типа:

В этом примере память выделяется под массив из четырех элементов типа int. К сожалению, данная форма оператора new не

```
int *pia = new int[4];
```

позволяет инициализировать элементы массива.

Некоторую путаницу вносит то, что обе формы оператора new возвращают одинаковый указатель, в нашем примере это указатель на целое. И pint, и pia объявлены совершенно одинаково, однако pint указывает на единственный объект типа int, а pia – на первый элемент массива из четырех объектов типа int.

Когда динамический объект больше не нужен, мы должны явным образом освободить отведенную под него память. Это делается с помощью оператора delete, имеющего, как и new, две формы – для единичного объекта и для массива:

```
delete pint; // освобождение единичного объекта
delete[] pia; // освобождение массива
```

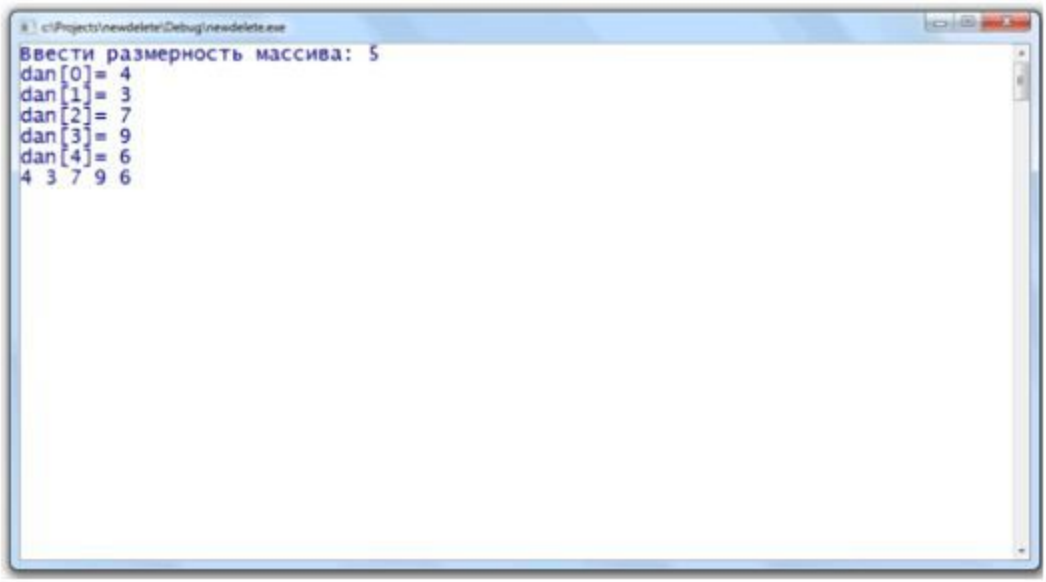
Что случится, если мы забудем освободить выделенную память? Память будет расходоваться впустую, она окажется неиспользуемой, однако возвратить ее системе нельзя, поскольку у нас нет указателя на нее. Такое явление получило специальное название *утечка памяти*. В конце концов программа аварийно завершится из-за нехватки памяти (если, конечно, она будет работать достаточно долго). Небольшая утечка трудно поддается обнаружению, но существуют утилиты, помогающие это сделать.

Пример: Создание динамического массива

```
1) #include <iostream>
2) using namespace std;
3) int main() {
4)    int size;
5)    int *dan;
6)    system("chcp 1251");
7)    system("cls");
8)    cout << "Ввести размерность массива: ";
9)    cin >> size;
10)    dan = new int[size];
```

```
11) for (int i=0; i<size; i++) {
12)     cout << "dan[" << i << "]= ";
13)     cin >> dan[i];
14)   }
15)   for (int i=0; i<size; i++)
16)     cout << dan[i] << " ";
17)   delete[] dan;
18)   cin.get(); cin.get();
19)   return 0;
20) }
```

Результат выполнения



ВАРИАНТЫ ЗАДАНИЙ

1. Объявите указатель на массив типа double и предложите пользователю выбрать его размер. Далее напишите четыре функции: первая должна выделить память для массива, вторая — заполнить ячейки данными, третья — показать данные на экран, четвертая — освободить занимаемую память. Программа должна предлагать пользователю продолжать работу (создавать новые

динамические ассивы) или выйти из программы.

2. Объявите указатель на массив типа int и выделите память память для 12-ти элементов. Необходимо написать функцию, которая поменяет значения четных и нечетных ячеек массива.
   Например, есть массив из 4-х целочисленных элементов:

	ячейка 0	ячейка 1	ячейка 2	ячейка 3
Исходные данные массива	1	2	3	4

	ячейка 0	ячейка 1	ячейка 2	ячейка 3
Данные после работы функции	2	1	4	3

3. Объявить и заполнить двумерный динамический массив случайными числами от 10 до 50. Показать его на экран. Для заполнения и показа на экран написать отдельные функции. (подсказка: функции должны принимать три параметра — указатель на динамический массив, количество строк, количество столбцов). Количество строк и столбцов выбирает пользователь.