ЛАБОРАТОРНАЯ РАБОТА № 4
УКАЗАТЕЛИ. ДИНАМИЧЕСКИЕ МАССИВЫ.

1. ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

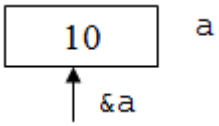
Целью работы является получение обучающимися навыков работы с динамической памятью в языке C++.

Задачами работы являются:

- написание программы на языке C++ с использованием указателей при обработке динамических массивов;
- тестирование работоспособности программы для различных исходных данных.

2. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

При обработке оператора определения переменной, например, `int a=10;` компилятор в памяти выделяет участок памяти в соответствии с типом переменной и записывается в этот участок указанное значение. Все обращения к переменной `a` компилятор заменит на адрес области памяти, в которой хранится эта переменная. Операция `&a` является операцией взятия адреса ее операнда.



Программист может определить собственные переменные для хранения адресов областей памяти.

Указатель – именованный объект, предназначенный для хранения адреса области памяти (объекта, непоименованной области оперативной памяти либо точки входа в функцию).

Указатель не является самостоятельным типом, он всегда связан с каким-то другим типом.

В общем случае синтаксис определения указателя на объект:
`Тип*Описатель;`

При определении указателя специфицируется имя указателя и тип объекта, на который он ссылается. Тип задает тип объекта, адрес которого будет содержать определяемая переменная и может соответствовать базовому, пустому (родовому, то есть типу `void`), перечислению, структурному типу и типу объединения. Реально указатель на `void` ни на что не указывает, но обладает способностью

указывать на область любого размера после его типизирования каким-либо объектом.

Описатель – это идентификатор, определяющий имя объявляемой переменной типа указатель или конструкция, которая организует непосредственно доступ к памяти. Описателю обязательно должна предшествовать звездочка (*).

Знак '*' является унарной операцией косвенной адресации, его операнд – указатель, а результат – адрес объекта, на который указывает операнд. Адресация является косвенной, так как обращение к области памяти осуществляется не напрямую по адресу (например, 1A2B), а через объект, которому в памяти соответствует определенный участок. Объем памяти, который выделяется для хранения данных, определяется типом данных и моделью памяти. Для приведенной на рисунке 2 модели памяти адресом переменной типа `float` с именем `summa` является `0012FF48`, адресом переменной типа `int` с именем `date` является `0012FF54`, адресом переменной типа `char` с именем `ch` является `0012FF63`.

Машинный адрес	0012FF48 байт	0012FF49 байт	0012FF4A байт	0012FF4B байт	0012FF54 байт	0012FF55 байт	0012FF56 байт	0012FF57 байт	0012FF63 байт
Значение в памяти	$2.015*10^{-6}$				1937				'G'
Имя	summa				date				ch

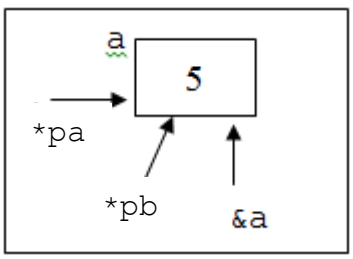
Способы инициализации указателя

1. Присваивание указателю адреса области памяти существующего объекта:
   - с помощью операции получения адреса:

```
int a=5;
int *pa=&a;
```

   - с помощью проинициализированного указателя

```
int *pb=pa;
```



2. Присваивание указателю адреса области памяти в явном виде:

```
char *cp=(char*)0x B8000000;
```

где `0xB8000000` – шестнадцатеричная константа, (`char*`) операция приведения типа (в практических задачах явное присваивание адреса указателю не используется).

3. Присваивание указателю пустого значения:

```
int *pN = NULL; или int *pN = 0;
```

С указателями можно выполнять следующие операции:

– разыменование (`*`) – получение значения величины, адрес которой хранится в указателе;

– взятие адреса (`&`);

– присваивание;

– арифметические операции

– сложение указателя только с константой,

– вычитание: допускается разность указателей и разность указателя и константы,

– инкремент (`++`) увеличивает значение указателя на величину `sizeof(тип);`

– декремент (`--`) уменьшает значение указателя на величину `sizeof(тип);`

– сравнение;

– приведение типов.

Для взятия адреса переменной также используется оператор &, размещаемый перед объектом, адрес которого хочется получить.

Пример:

```
int a = 15;
cout << &a << endl; // выводится адрес переменной, а не её значение
```

Для перехода по известному адресу используется оператор `*`, размещаемый перед адресом или указателем хранящем адрес. Под переходом по адресу понимается, что от адреса мы переходим к

действиям над значением, хранимом по данному адресу. Это операция называется иногда разыменованием.

```
    int a = 15; // переменная a со значением
    int* p = &a; // указатель с адресом переменной
a
    cout << *p << endl; // увидим 15, т.е.
значение переменной
```

Таким образом, к любой переменной можно обратиться как по её имени, так и по её адресу, применив к нему операцию разыменования. Справедливо тождество: выражение `a == *(&a)`, где `a` – любого типа.

Существует два основных способа хранения информации в оперативной памяти. Первый заключается в использовании глобальных и локальных переменных. В случае глобальных переменных выделяемые под них поля памяти остаются неизменными на все время выполнения программы. Под локальные переменные программа отводит память из стекового пространства. Однако локальные переменные требуют предварительного определения объема памяти, выделяемой для каждой ситуации. Хотя C++ эффективно реализует такие переменные, они требуют от программиста заранее знать, какое количество памяти необходимо для каждой ситуации.

Второй способ, которым C++ может хранить информацию, заключается в использовании системы динамического распределения. При этом способе память распределяется для информации из свободной области памяти по мере необходимости. Область свободной памяти находится между кодом программы с ее постоянной областью памяти и стеком.

Динамическое распределение памяти – способ выделения оперативной памяти компьютера для объектов в программе, при котором выделение памяти под объект осуществляется во время выполнения программы.

При динамическом распределении памяти объекты размещаются в т. н. «куче» (англ. heap): при конструировании объекта указывается размер запрашиваемой под объект памяти, и, в случае успеха, выделенная область памяти, условно говоря, «изымается» из «кучи», становясь недоступной при последующих операциях выделения памяти. Противоположная по смыслу операция – освобождение занятой ранее под какой-либо объект памяти: освобождаемая память, также условно говоря, возвращается в «кучу» и становится доступной при дальнейших операциях выделения памяти.

В языке программирования C++ для динамического распределения памяти существуют операции `new` и `delete`. Эти операции используются для выделения и освобождения блоков памяти. Область памяти, в которой размещаются эти блоки, называется свободной памятью.

Операция `new` позволяет выделить и сделать доступным свободный участок в основной памяти, размеры которого соответствуют типу данных, определяемому именем типа.

Синтаксис:

```
new ИмяТипа;
```

Операцию `delete` следует использовать только для указателей на память, выделенную с помощью операции `new`.

Пример.

```
#include <iostream>
using namespace std;
int main(){
  int *pa, *pb;
  pa = new int; // выделяется 4 Б из «кучи»
  *pa = 21;
  pb = pa;
  cout << *pa << "\t" << *pb << "\t" << pa << "\t"
<< pb << endl;
  pb = new int; // выделяется 4 Б из «кучи»
  *pb = 28;
```

```
  cout << *pa << "\t" << *pb << "\t" << pa << "\t"
<< pb << endl;
  delete pb;
  cout << *pa << "\t" << *pb << "\t" << pa << "\t"
<< pb << endl;
  delete pa;
//  system("pause");
  return 0;
}
```

Связь с массивами

При создании любого массива в C++, вместе с ним естественным образом создаётся константный указатель. Имя этого указателя совпадает с именем массива. Тип этого указателя – «указатель на базовый тип массива». В появившемся указателе хранится адрес начального элемента массива.

С учётом того, что в массиве все элементы располагаются в памяти последовательно, начав с указателя, направленного на начальный элемент, мы сможем обойти все элементы массива, смещая указатель на каждом шаге вправо на минимально возможную дистанцию (то есть, на соседний справа элемент подходящего типа). Смещение указателя может производиться с помощью операторов инкремента и декремента.

Пример.

```
int ar[] = {-72, 3, 402, -1, 55, 132};
int* p = ar;
for (int i=0; i<=5; i++) {
 cout << *p << ' ';
 p++;
}
for (int i=0; i<=5; i++) {
  cout << &ar[i] << ' ';
}
```

В примере все элементы массива будут выведены на экран без использования индексов. Далее выведены адреса элементов массива. Они будут отличаться на размер в байтах базового типа массива (в данном случае, `int`).

Справедливо тождество: выражение `a[i] == *(a+i)`, где `a` указатель на массив любого типа и `i` допустимый индекс этого массива.

Динамические массивы.

Динамический массив – это массив, размер которого заранее не фиксирован и может меняться во время исполнения программы. Для изменения размера динамического массива язык программирования C++, поддерживающий такие массивы, предоставляет специальные встроенные функции или операции. Динамические массивы дают возможность более гибкой работы с данными, так как позволяют не прогнозировать хранимые объемы данных, а регулировать размер массива в соответствии с реально необходимыми объемами.

Под объявлением одномерного динамического массива понимают объявление указателя на переменную заданного типа для того, чтобы данную переменную можно было использовать как динамический массив.

Описание.

```
Тип * ИмяМассива;
```

Таким образом, при динамическом распределении памяти для динамических массивов следует описать соответствующий указатель, которому будет присвоено значение адреса начала области выделенной памяти.

Выделение памяти под одномерный динамический массив производится при помощи операции `new`, которая выделяет для размещения массива участок динамической памяти соответствующего размера и не позволяет инициализировать элементы массива.

Описание.

```
ИмяМассива = new Тип [Размер];
```

Освобождение памяти, выделенной под одномерный динамический массив осуществляется при помощи операции `delete`, которая освобождает участок памяти ранее выделенной операцией `new`.

Описание.

```
delete [] ИмяМассива;
```

Пример. Найти сумму элементов динамического целочисленного массива. Все отрицательные элементы заменить нулями, при их отсутствии – вывести сообщение.

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <locale.h>

using namespace std;

int main()
{
    setlocale(0, "");
    srand(time(NULL));
    int i, N;
    cout <<  "Введите размер массива: ";
    cin >> N;

    //объявление одномерного динамического
массива
    int *A = new int[N];
    // Генерация элементов массива
    for (i = 0; i < N; i++)
    {
        *(A+i) = rand() % 101 - 50;
    }

    // Вывод элементов исходного массива
    cout << endl <<  "Исходный массив" <<
endl;
    for (i = 0; i < N; i++)
    {
        cout << *(A+i) << " ";
    }
    cout << endl

    // Вычисление суммы элементов массива
    int summ = 0;
```

```
        for (i = 0; i < N; i++)
        {
            summ += *(A+i);
        }
        cout <<  "Сумма элементов массива = " <<
summ << endl;

        // Обработка массива
        bool flag = false; // наличие
отрицательных элементов
        for (i = 0; i < N; i++)
        {
            if (*(A+i) < 0){
                *(A+i) = 0;
                flag = true; // отрицательные
элементы найдены
            }
        }

        // Вывод элементов результирующего массива
        if (flag){
                cout << endl <<  "Результирующий
массив" << endl;
                for ( i = 0; i < N; i++){
                    cout << *(A+i) << " ";
                    }
                cout<<endl;
                }
            else
                cout << "Отрицательые элементы не
обнаружены" << endl;
        delete []A;
        return 0;
    }
```

3. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Получение варианта задания на лабораторную работу.
2. Разработка алгоритма и графической схемы решения задачи.

3. Составление программы на языке C++.
4. Отладка программы.
5. Тестирование программы.
6. Составление отчета о проделанной работе.

4. ТРЕБОВАНИЯ К ОФОРМЛЕНИЮ ОТЧЕТА

1. Оформить титульный лист с указанием темы работы.
2. Сформулировать цель и задачи работы.
3. Привести формулировку задания.
4. Построить графическую схему алгоритма.
5. Записать текст программы.
6. Привести результаты тестирования программы.
7. Сформулировать вывод по проделанной работе.

5. ВАРИАНТЫ ЗАДАНИЙ

1. Решить задачу для динамического массива, не используя индексацию. Дан массив размера N. Найти среднее арифметическое отрицательных элементов массива. Вывести адреса четных элементов исходного массива.
2. Решить задачу для динамического массива, не используя индексацию. Проверить, образуют ли элементы целочисленного массива размера N арифметическую прогрессию. Если да, то вывести разность прогрессии, если нет – вывести сообщение. Вывести адреса отрицательных элементов исходного массива.
3. Решить задачу для динамического массива, не используя индексацию. Дан массив ненулевых целых чисел размера N. Проверить, чередуются ли в нем положительные и отрицательные числа. Если чередуются, то вывести сообщение, если нет, то вывести номер первого элемента, нарушающего закономерность. Вывести адрес минимального элемента исходного массива.
4. Решить задачу для динамического массива, не используя индексацию. Заменить все положительные элементы целочисленного массива, состоящего из N элементов, на значение максимального элемента. Вывести адреса нечетных элементов исходного массива.
5. Решить задачу для динамического массива, не используя индексацию. Расположить элементы данного массива в обратном порядке (первый элемент меняется с последним,

второй – с предпоследним и т.д. до среднего элемента). Вывести адреса k последних элементов исходного массива.

6. Решить задачу для динамического массива, не используя индексацию. В заданном одномерном массиве максимальный элемент заменить значением суммы предшествующих ему элементов. Вывести адреса отрицательных элементов исходного массива.
7. Решить задачу для динамического массива, не используя индексацию. В данном массиве поменять местами элементы, стоящие на нечетных местах, с элементами, стоящими на четных местах. Вывести адреса нулевых элементов исходного массива.
8. Решить задачу для динамического массива, не используя индексацию. Найти сумму нечетных элементов, стоящих на нечетных местах (то есть имеющих нечетные номера). Вывести адрес максимального элемента исходного массива.
9. Решить задачу для динамического массива, не используя индексацию. Дан одномерный целочисленный массив. Необходимо “сжать” массив, удалив из него каждый второй элемент, а оставшиеся элементы заменить нулями. Вывести адреса k первых элементов исходного массива.
10. Решить задачу для динамического массива, не используя индексацию. Дан целочисленный массив и числа a и b. Переместить в начало массива элементы, находящиеся в промежутке $[a, b]$. Вывести адрес минимального элемента исходного массива.
11. Решить задачу для динамического массива, не используя индексацию. Задан массив с количеством элементов N. Сформируйте два массива: в первый включите элементы исходного массива с четными номерами, а во второй – с нечетными. Вывести адреса элементов исходного массива, больших M.
12. Решить задачу для динамического массива, не используя индексацию. Дана последовательность N целых чисел и целое число m. Указать пары чисел этой последовательности таких, что их сумма равна m. Вывести адреса нулевых элементов исходного массива.

13. Решить задачу для динамического массива, не используя индексацию. Дано N чисел. Наименьший член этой последовательности заменить значением среднего арифметического всех элементов, остальные элементы оставить без изменения. Вывести адреса четных элементов исходного массива.
14. Решить задачу для динамического массива, не используя индексацию. Дана последовательность из N различных целых чисел. Найти сумму ее членов, расположенных между максимальным и минимальным значениями (в сумму включить и оба этих числа). Вывести адреса отрицательных элементов исходного массива.
15. Решить задачу для динамического массива, не используя индексацию. Дан целочисленный массив размера N. Преобразовать его, прибавив к отрицательным числам последний элемент. Первый и последний элементы массива не изменять. Вывести адрес минимального элемента исходного массива.
16. Решить задачу для динамического массива, не используя индексацию. Найти сумму всех четных элементов массива, стоящих на четных местах, то есть имеющих четные номера. Вывести адреса k последних элементов исходного массива.
17. Решить задачу для динамического массива, не используя индексацию. В одномерном массиве заменить нулями все положительные элементы, идущие после минимального элемента массива. Вывести адреса элементов исходного массива, расположенных внутри интервала $[a, b]$.
18. Решить задачу для динамического массива, не используя индексацию. В заданном массиве определить среднее арифметическое значение положительных и среднее арифметическое значение отрицательных элементов. Вывести адреса нечетных элементов исходного массива.
19. Решить задачу для динамического массива, не используя индексацию. Даны два одномерных целочисленных массива, состоящие из одинакового числа элементов. Получить третий массив той же размерности, каждый элемент которого равен большему из соответствующих элементов данных массивов. Вывести адреса нулевых элементов исходного массива.

20. Решить задачу для динамического массива, не используя индексацию. Дан целочисленный массив A размера N. Вывести номера тех его элементов, которые больше первого и меньше последнего элементов. Если таких элементов нет, то вывести сообщение. Вывести адреса k первых элементов исходного массива.
21. Решить задачу для динамического массива, не используя индексацию. Дан целочисленный массив размера N. Преобразовать его, прибавив к четным числам первый элемент. Первый и последний элементы массива не изменять. Вывести адрес максимального элемента массива.
22. Найти среднее геометрическое положительных элементов одномерного вещественного массива. Вывести адреса элементов исходного массива, расположенных внутри интервала $[a, b]$.
23. Решить задачу для динамического массива, не используя индексацию. Дан целочисленный массив размера N. Вывести сначала все его элементы, являющиеся четными числами, а затем – нечетными. Вывести адреса элементов исходного массива, расположенных вне интервала $[a, b]$.
24. Решить задачу для динамического массива, не используя индексацию. Поменять местами минимальный и максимальный элементы одномерного массива размера N. Вывести адрес минимального элемента исходного массива.
25. Решить задачу для динамического массива, не используя индексацию. Дан массив, состоящий из N элементов. Переставить в обратном порядке элементы массива, расположенные между его минимальным и максимальным элементами. Вывести адреса элементов исходного массива, больших M.

6. КРИТЕРИИ РЕЗУЛЬТАТИВНОСТИ ВЫПОЛНЕНИЯ РАБОТЫ

Лабораторная работа считается успешно выполненной, если:

- представленный отчет содержит все необходимые пункты, согласно требованиям;

- успешно демонстрируется работа программы как на выбранных студентом исходных данных, так и на заданных преподавателем исходных данных;
- студент правильно отвечает на контрольные вопросы.

7. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое указатель, динамическая память?
2. Всегда ли эффективно использование динамической памяти в программах?
3. Почему ошибки при работе с динамической памятью относят к опасным?
4. Опишите последствия использования в программах неинициализированных указателей.
5. Почему выделенную ранее динамическую память следует освобождать после использования?
6. К каким последствиям в работе программы приводит попытка освободить динамическую память, не выделенную ранее?
7. Почему к нулевому указателю нельзя применить операцию разыменования?
8. Какая связь между массивом и указателем на его нулевой элемент?