Лабораторная работа №7
Связь массивов и указателей

1. Цель работы:

1) Получение практических навыков при работе с указателями.
2) Использование указателей при работе с массивами.

.

2. Теоретические сведения

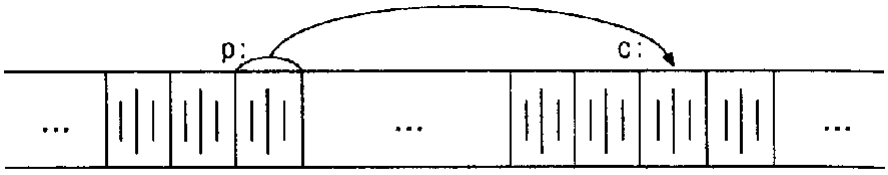
Массив – это упорядоченная последовательность переменных одного типа. Каждому элементу массива отводится одна ячейка памяти. Элементы одного массива занимают последовательно расположенные ячейки памяти. Все элементы имеют одно имя – имя массива и отличаются индексами – порядковыми номерами в массиве. Количество элементов в массиве называется его размером.

2.1 Указатели и массивы

1. Указатели и адреса

Указатели являются специальными объектами в программах на C/C++. Указатели предназначены для хранения адресов памяти.
Использование указателей делает программы обычно короче и эффективнее.
Рассмотрим упрощенную схему организации памяти. Память типичной машины подставляет собой массив последовательно пронумерованных или проадресованных ячеек, с которыми можно работать по отдельности или связными кусками

Если *p* - указатель на *c*, то ситуация выглядит следующим образом:



Унарный оператор **&** выдает адрес объекта, так что инструкция

```
p = &c;
```

присваивает переменной *p* адрес ячейки *c* (говорят, что *p* указывает на *c*)

Унарный оператор ***** есть оператор *косвенного доступа*. Примененный к указателю он выдает объект, на который данный указатель указывает.

Знак `*`, обозначает указатель и относится к типу переменной, поэтому его рекомендуется ставить рядом с типом, а от имени переменной отделять пробелом, за исключением тех случаев, когда описываются несколько указателей. При описании нескольких указателей знак `*` ставится перед именем переменной-указателя, т. к. иначе будет не понятно, что эта переменная также является указателем.

Примеры:

```
int* i;
double *f, *ff;//два указателя
char* c;
```

Указатель может быть константой или переменной, а также указывать на константу или переменную.

```
int i;                  //целая переменная
const int ci=1;         //целая константа
int* pi;                //указатель на целую переменную
const int* pci;         //указатель на целую константу
```

Указатель можно сразу проинициализировать:

```
//указатель на целую переменную
int* pi=&i;
```

Унарные операторы ***** и **&** имеют более высокий приоритет, чем арифметические операторы, так что присваивание

```
y = *ip + 1;
```

берет то, на что указывает *ip*, и добавляет к нему 1, а результат присваивает переменной *y*. Аналогично

```
*ip += 1;
```

увеличивает на единицу то, на что указывает *ip*; те же действия выполняют

```
++*ip;
```

и

```
(*ip)++;
```

В последней записи скобки необходимы, поскольку если их не будет, увеличится значение самого указателя, а не то, на что он указывает. Это обусловлено тем, что унарные операторы ***** и **++** имеют одинаковый приоритет и порядок выполнения - справа налево.

И наконец, так как указатели сами являются переменными, в тексте они могут встречаться и без оператора косвенного доступа. Например, если *iq* есть другой указатель на *int*, то

```
iq = ip;
```

копирует содержимое *ip* в *iq*, чтобы *ip* и *iq* указывали на один и тот же объект.

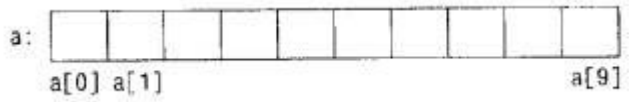
2.Указатели и одномерные массивы

В C/C++ существует связь между указателями и массивами. Любой доступ к элементу массива, осуществляемый операцией индексирования, может быть выполнен с помощью указателя. Вариант с указателями в общем случае работает быстрее.

Объявление

```
int a[10];
```

Определяет массив *a* размера 10, т. е. блок из 10 последовательных объектов с именами a[0], a[1], ..., a[9].



Запись *a[i]* отсылает нас к *i*-му элементу массива. Если *pa* есть указатель на *int*, т. е. объявлен как

```
int *pa;
```

то в результате присваивания

```
pa = &a[0];
```

pa будет указывать на нулевой элемент *a*, иначе говоря, *pa* будет содержать адрес элемента *a[0]*.

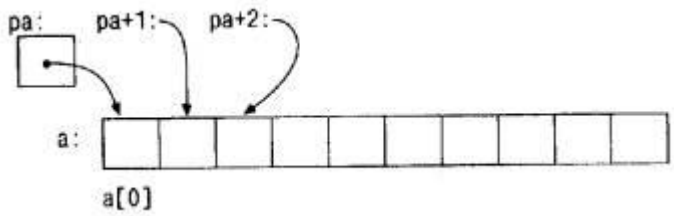
Теперь присваивание

```
x = *pa;
```

будет копировать содержимое *a[0]* в *x*.

Если *pa* указывает на некоторый элемент массива, то *pa+1* по определению указывает на следующий элемент, *pa+i* - на *i*-й элемент после *pa*

Таким образом, если *pa* указывает на *a[0]*, то **(pa+i)* - содержимое *a[i]*.



Таким образом, после присваивания

```
pa = &a[0];
```

pa и *a* имеют одно и то же значение. Поскольку имя массива является синонимом расположения его начального элемента, присваивание *pa=&a[0]* можно также записать в следующем виде:

```
pa = a;
```

Вычисляя *a[i]*, Си сразу преобразует его в **(a+i)*; указанные две формы записи эквивалентны. Из этого следует, чтозаписи *&a[i]* и *a+i* также будут эквивалентными.

С другой стороны, если *pa* - указатель, то его можно использовать с индексом, т. е. запись *pa[i]* (массива с индексом) эквивалентна записи **(pa+i)(указатель со смещением)*

Различие: *Указатель - это переменная*, поэтому можно написать *pa=a* или *pa++*. Но *имя массива не является переменной*, и записи вроде *a=pa* или *a++* не допускаются.

3. Адресная арифметика

- Если *p* есть указатель на некоторый элемент массива, то *p++* увеличивает *p* так, чтобы он указывал на следующий элемент, а *p+=i* увеличивает его, чтобы он указывал на *i*-й элемент после того, на который указывал ранее. Эти и подобные конструкции - самые простые примеры арифметики над указателями, называемой также адресной арифметикой.
- Если *p* и *q* указывают на элементы одного массива, то к ним можно применять операторы отношения ==, **!**=, <, >= и т. д. Например, отношение вида

  ```
  p < q
  ```

  истинно, если *p* указывает на более ранний элемент массива, чем *q*. Любой указатель всегда можно сравнить на равенство и неравенство с нулем. А вот для указателей, не указывающих на элементы одного массива, результат арифметических операций или сравнений не определен.

- Указатели и целые можно складывать и вычитать. Конструкция

  ```
  p + n
  ```

  означает адрес объекта, занимающего *n*-е место после объекта, на который указывает *p*. Это справедливо безотносительно к типу объекта, на который указывает *p*; *n* автоматически домножается на коэффициент, соответствующий размеру объекта. Информация о размере неявно присутствует в объявлении *p*. Если, к примеру, *int* занимает четыре байта, то коэффициент умножения будет равен четырем.

- Допускается также вычитание указателей. Например, если *p* и *q* указывают на элементы одного массива и *p<q*, то *q-p+1* есть число элементов от *p* до *q* включительно. Этим фактом можно воспользоваться при написании еще одной версии *strlen*:

```
int strlen(char *s)  /* strlen: возвращает длину строки s */
{char *p = s;
   while (*p != '\0')
      p++;
   return p - s;
}
```

В начале *p* указывает на первый символ строки. На каждом шаге цикла *while* проверяется очередной символ; цикл продолжается до тех пор, пока не встретится '\0'. Каждое продвижение указателя *p* на следующий символ выполняется инструкцией *p++*, и разность *p-s* дает число пройденных символов, т. е. длину строки.

Нельзя складывать два указателя, перемножать их, делить, сдвигать, выделять разряды; указатель нельзя складывать со значением типа *float* или *double*; указателю одного типа нельзя даже присвоить указатель другого типа, не выполнив предварительно операции приведения (исключение - указатели типа void*).

Перейдем к примерам:

Пример 1. Рассмотрим статичный одномерный массив определенной длинны и инициализируем его элементы

```
void main(){
        const int size = 7;     // объявление
        int  array[size];
                for (int i = 0; i != size; i++)    // инициализация элементов массива
                array[i] = i*i;
        }
```

А теперь будем обращаться к элементам массива, используя указатели:

```
int* arr_ptr = array;  // инициализируем указатель arr_ptr адресом начала массива array.
        for (int i = 0; i != size; i++)
                cout << *(arr_ptr + i) << endl;     // или cout<<*arr++ (1*)
```

Использование идентификатора массива без указания квадратных скобок эквивалентно указанию адреса его первого элемента В цикле мы выводим элементы, обращаясь к каждому с помощью начального адреса и смещения. То есть:

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

*(arr_ptr + 1) — первый (*i* = 1), и так далее.

Пример 2. Теперь в указатель «явно» занесем адрес первого элемента массива.

```
int* arr_ptr_null = &array[0];
        for (int i = 0; i != size; i++){
                cout << *(arr_ptr_null + i) << endl;
```

Пройдем по элементам с конца массива:

```
int* arr_ptr_end = &array[size - 1];
        for (int i = 0; i != size; i++)
                cout << *(arr_ptr_end - i) << endl;
```

2.2. Динамическое выделение памяти

1. Использование указателей для одномерных динамических массивов.

Зачастую при решении какой-либо задачи возникает потребность в использовании массива неопределенного размера, то есть размер этот заранее неизвестен. На помощь приходят динамические массивы — память под них выделяется в процессе выполнения программы.

Пример 3:

```
cin>> size;
int* dyn_arr = new int[size];
```

Здесь объявляем указатель и инициализируем его началом массива, под который выделяется память оператором **new** на *size* элементов. В этом случае мы можем использовать те же приемы в работе с указателями, что и с статическим массивом.

Что следует из этого извлечь — если вам нужна какая — то структура (как массив, например), но ее размер вам заранее неизвестен, то просто сделайте объявление этой структуры, а проинициализируете ее уж позже.

Здесь имеют место логические скобки – выделивши память (ресурс), мы теперь за нее отвечаем – то есть, должны ее вернуть на место. Закрывающая логическая скобка для **new** – оператор **delete**. Работа с динамической памятью выглядит так:

```
int * pAr = new int[Size];
// работа
delete [] pAr;
//|========
int * pA = new int;
// работа
delete pA;
```

То есть, если мы выделяем массив используя оператор **new** со скобками **[]**, мы и вернуть его должны как масссив – используя **delete** со скобками **[]**, иначе программа неправильно вернет блок памяти и будет дальше работать неадекватно. Нужно всегда использовать соответствующий оператор. **new** и **new[]**– это совершенно разные и не взаимозаменяемые операторы, точно так же как и **delete** и **delete[]**.

2.Двойные указатели.

Для инициализации двумерного динамического массива используется *указатель на указатель*. Это та же переменная, которая хранит адрес другого указателя „более низкого порядка“.

Пример 4:

```
Cin>>size;      // двумерный массив размером size*size
int** arr = new int*[size];   // выделение памяти под массив указателей
for(int i = 0; i != size; i++){
    arr[i] = new int[size];  //выделение памяти для массива значений
}
.....
for(int i = 0; i != size; i++)
delete arr[i];  // освобождение памяти из под массива значений


delete[] arr;  // освобождение памяти из под массива указателей
```

3. Лабораторное задание

3.1 Практическая работа

3.1.1. Проверьте результат работы **Примера 1** для динамического выделения памяти, заполнив элементы массива генератором случайных чисел, используя **(1*)**

3.1.2. Протестируйте приведенную ниже программу и объясните результаты ее работы.

```
#include <iostream>

#include <stdlib.h>

#include <ctime>

using namespace std;

int main(int argc, char** argv)

{ int N,m,i; float  *pt;

srand(time(NULL));

cin>>N;

float* A= new float[N];

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

   { A[i] =( float) rand()/ RAND_MAX;

    cout<<*(A+i)<<"\n ";

   }

pt=A;

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

 { A[i] = A[i] * 2;

   cout<<*(pt+i)<<" ";}

cout<<endl;

cout<<*(pt+1)<<" "<<*pt+1<<' '<<pt<<' '<<*pt<<endl;

            for ( i = N-1; i >= 0; i -- )

                              cout<<*(A+i)<<"\n ";

 return 0;

}
```

3.2 Индивидуальное задание по вариантам

1) Дан массив A[N]. Элементы массива заполнить, используя функцию генератора случайных чисел: целого типа - для вариантов с четными номерами заданий, вещественного типа – для вариантов с нечетными номерами.Составить программу обработки одномерного массива. В программе необходимо определить указатели. Все обращения к элементам массивов производить с помощью указателей. Вывести полученные результаты. (При вводе/выводе элементов можно использовать индексы)

    1. Написать программу, уменьшающую все положительные элементы на значение минимального элемента.
    2. Написать программу, меняющую местами наибольший элемент одномерного массива с последним.
    3. Написать программу, меняющую в одномерном массиве местами первый элемент с минимальным.
    4. Написать программу, определяющую разность между суммой модулей отрицательных элементов и суммой положительных элементов одномерного массива.
    5. Написать программу, заменяющую все элементы одномерного массива, кроме максимального, на их отрицательные значения.
    6. Написать программу, определяющую минимальный элемент в одномерном массиве и выводящую его адрес.
    7. Написать программу, заменяющую положительные элементы массива на их квадраты.
    8. Написать программу, определяющую минимальный элемент в одномерном массиве и увеличивающую его в два раза.
    9. Написать программу, определяющую модуль разности между количеством отрицательных и положительных элементов одномерного массива.
    10. Написать программу, определяющую количество отрицательных элементов в массиве и выводящую их адреса.
    11. Написать программу, определяющую среднее значение положительных элементов одномерного массива.
    12. Написать программу, уменьшающую все элементы на среднее значение положительных элементов одномерного массива.
    13. Написать программу, определяющую максимальное квадратичное отклонение элементов одномерного массива от среднего значения.
    14. Написать программу вычисления знакопеременной суммы элементов массива: $a_1 – a_2 + a_3 – a_4 + ...$
    15. Написать программу, заменяющую все элементы после максимального элемента на противоположные.
    16. Ввести четное количество символов. Вывести вторую половину их в обратной последовательности
    17. Ввести четное количество символов. Поменять местами каждую пару элементов массива. Вывести результат.
    18. Сформировать массив B(10) из массива A(10) по следующему принципу: B(N)=A(N)+N.
    19. Сформировать массив B(5) из элементов первой половины массива A(10) в обратной последовательности.
    20. Написать программу, заменяющую все элементы, меньшие чем среднее арифметическое массива, нулями.
    21. Сформировать массив B(5) из элементов второй половины массива A(10) в обратной последовательности.
    22. Сформировать массив B(10) из квадратов элементов массива A(10) в обратной последовательности.