Арифметика указателей. Связь с массивами.

В C++ разрешены некоторые арифметические операции с указателями. Однако, все такие операции неявно предполагают, что все участвующие в них указатели (как исходные и промежуточные результаты, так и окончательные) указывают на элементы одного и того же массива. Однако, компилятор никак не проверяет истинность этого предположения и соблюдение его всецело на совести программиста.

Если указатель p указывает на элемент массива с индексом i, то p+j, где j — целое число, будет указывать на элемент того же массива с индексом i+j. При этом совершенно неважно, положительно j или отрицательно, лишь бы окончательный индекс лежал в допустимых пределах; аналогичным образом, p-j указывает на элемент с индексом i-j. По тем же правилам работают и операции инкремента и декремента. Также, если указатели p и q указывают на элементы с индексами i и j, то выражение p-q имеет целый результат и равно i-j (опять же неважно, положительно это значение, равно нулю или отрицательно).

Этим, однако, арифметика указателей и ограничивается. Несмотря на то, что указатели, с точки зрения внутреннего устройства, это просто адреса в памяти (целые числа), тем не менее их нельзя складывать или умножать друг на друга или даже на целые числа. Если такая нужда возникает (и Вы знаете, что делаете), можно явно превратить указатель в целое число при помощи операции преобразования типов, написав перед ним (int). Затем можно проделать необходимые манипуляции с полученным целым числом, и наконец, снова превратить теперь уже целое число обратно в указатель при помощи той же операции преобразования типов, написав перед ним (тип *), где тип — тип той переменной, в указатель на которую Вы хотите превратить полученное целое число.

Заметим здесь еще следующее. Во-первых, в большинстве выражений, в которых участвует имя массива, оно понимается как указатель на первый элемент этого массива (с индексом 0). Во-вторых, можно индексировать указатели так же, как и массивы: запись p[i] означает то же самое, что и *(p+i). Однако, это работает только для одномерных массивов: имя двумерного массива понимается как указатель на первую строку (одномерный массив), и не приводится к типу «указатель на указатель». Также, из этого правила имеются очевидные исключения: в выражении sizeof M, если M — массив (неважно, одномерный или многомерный), вычисляется размер массива M, а не указателя.

Напоследок скажем только, что арифметика указателей не работает для указателей неизвестно на что (типа void *).

Задачи.

1. Объявить последовательно переменные типов `int`, `char`, `bool`, `short`, `long`, `float`, `double`, `long double`, `int`. Вывести их адреса; сравнить разности адресов с размерами переменных, выдаваемыми операцией `sizeof`.

2. Написать функцию, принимающую массив целых чисел, его длину и указатель на какой-то элемент этого массива. Под шагом понимается сдвиг указателя на число элементов, равное значению указываемого элемента массива. Функция должна возвращать указатель на элемент массива-параметра, полученный при помощи 10 шагов из исходного указателя; при этом, если на каком-то шаге результат выходит за пределы массива, нужно возвращать последнее значение указателя, еще указывающее в пределах массива-параметра.

3. Написать функцию, принимающую два указателя на `float` (указывающие на элементы одного и того же массива), и возвращающую указатель, делящий интервал между ними (приблизительно) в отношении 3:5.

4. Написать функцию, принимающую 5 указателей на вещественные переменные (в пределах одного массива) и возвращающую такой указатель на элемент того же массива, что сумма модулей разностей между этим указателем и указателями-параметрами минимальна.

Мы уже говорили о том, что можно преобразовать тип указателя в целое число и обратно. Оказывается, принудительно можно преобразовать любой тип указателя в любой другой. Это приведет к тому, что содержимое памяти по этому адресу начнет трактоваться по-другому. Опреация преобразования типов указателей никак не меняет содержимого памяти по данному адресу; она лишь меняет способ трактовки этого содержимого, который тоже может быть важен. Например, и число типа int, и число типа float занимают по 4 байта. Но способ представления конкретных чисел, например 5 для int и 5.0 для float, существенно отличается один от другого. Поэтому следующий фрагмент выдаст совсем не 5.0, а нечто другое (что именно?):

```
int x = 5;
cout<<*(float*)&x<<endl;
```

Однако, такое преобразование переменных из одного типа в другой, сохраняющее не значение переменной, а содержимое памяти по данному адресу, иногда бывает важно. Например, таким образом можно превратить какое-либо значение в набор байтов, чтобы потом обрабатывать эти байты по-отдельности, скажем, для записи значения в файл в двоичном виде или передачи значения по сети.

Кроме обсуждаемого выше явного преобразования типов указателей имеется и неявное, которое происходит автоматически, без всяких усилий программиста. Это имеет место в следующих случаях:

1) Целое значение 0 (явно выписанное, и только оно) преобразуется в указатель любого типа, и означает указатель, никуда не указывающий.

2) Любой указатель может быть преобразован в тип void *, т. е. любой указатель можно присвоить переменной типа void * (обратное преобразование возможно только явно).

3) Любой указатель на один тип может быть преобразован в указатель на другой тип, который является синонимом первого, созданным при помощи конструкции typedef.

4) Наконец, при наследовании указатель на объект производного класса может быть преобразован в указатель на объект базового класса (обратное преобразование, как и в п. 2, возможно только явно).

Задачи.

1. Процессор называется Big Endian, если целые числа, занимающие несколько байтов, хранятся по следующему правилу: младший байт записан по наибольшему адресу, и Little Endian, если младший байт хранится по наименьшему адресу; если же порядок записи байтов в составе целого числа не совпадает ни с одним из упомянутых, назовем это Mixed Endian. Определить тип процессора, на котором мы работаем.

2. Написать макроопределение, выдающее значение последнего байта той переменной, на которую указывает параметр (это, естественно, указатель — больше о нем ничего не известно). Затем переделать это макроопределение в шаблонную функцию.

3. Написать функцию, принимающую указатель на double и 2 целых числа типа int, и записывающую в ту переменную, на которую указывает указатель-параметр, число типа double, составленное из двух целых чисел-параметров (число типа double занимает 8 байтов; первое целое число должно занимать байты с 0 по 3, второе — с 4 по 7).

4. Написать функцию, принимающую указатель неизвестно на что, и возвращающую другой указатель неизвесто на что, целочисленное значение которого равно $x^2 + 3x + 1$, где x — целочисленное значение указателя-параметра.

Что нужно знать в результате (проверяется на зачете):

- Какие арифметические операции можно выполнять над указателями и каков будет их результат?
- Какая связь существует между одномерными массивами и указателями?
- Почему арифметика указателей не работает для указателей неизвестно на что?
- Как изменить способ трактовки содержимого памяти, занимаемой какой-то переменной?
- Когда один тип указателя может быть автоматически преобразован в другой?
- Как получить адрес, содержащийся в переменной типа указателя?
- Как превратить целое число в адрес?

Что нужно уметь в результате (задача на контрольной работе):

- Пользоваться арифметикой указателей.
- Превращать один тип указателя в другой, указатель в целое число и обратно.