Практическое задание 10
Указатели в C++: практическое использование
Указатели и функции, указатели и массивы

Цель. Знакомство с указателями, как типом данных. Практическое использование указателей: использование указателей при передаче параметров в функцию, соотношение между указателями и массивами. Введение в динамические массивы. Разработка алгоритмов обработки массивов в функциях.

ПЛАН

1. Краткое теоретическое введение. Указатель как конструируемый тип 1
2. Массивы в статической памяти 2
3. Динамические массивы 2
4. Функции работы с массивами 2

1. Краткое теоретическое введение. Указатель как конструируемый тип

Указатели – программные объекты, значением которых являются адреса других объектов или области памяти. Назначение указателей – доступ к объектам или областям памяти напрямую.

Оперативная память, это поток адресуемых байт: адресов, пронумерованных в 16-ричной системе. Про объявлении любого объекта в коде, ему распределяется память в количестве, необходимом для хранения данного этого типа, и этот адрес сопоставлен имени объекта в коде.

Пример. Переменная типа int занимает 4 байта.

```
int A = 5; // sizeof(int) – 4 байта.
```

0x0136fe70	0x0136fe71	0x0136fe72	0x0136fe73
1-й байт	2-й байт	3-й байт	4-й байт
A = 5			

Операции адресной армфметики

Помимо специальных операций, предназначенных для того, чтобы можно было узнать значение адреса (операция & – получить адрес) или узнать значение, лежащее по адресу (унарная операция * - косвенная адресация), используются операции адресной арифметики.

Унарные ++ и -- еще называют операциями смещения указателя. Они изменяют значение адреса в зависимости от типа данных, с которым связан указатель, например, для `int` – на 4 байта, для `double` – на 8 байт. Мспользуются для косвенной адресации.

Массивы и указатели

Массив, это множество данных одного типа, для которых в оперативной памяти процесса распределяется непрерывная область адресного пространства, адресуемая именем массива.

Синтаксис описания массива:

```
Тип Имя_массива [Кол_во_элементов]; // Список инициализации
```

`Кол_во_элементов` – константная величина, то есть константа в чистом виде.

Семантика описания массива: компилятор распределяет память для хранения элементов массива в указанном количестве. Имя_массива сопоставлено всей совокупности данных и адресует первый байт в этой области, следовательно, имя массива, это указатель.

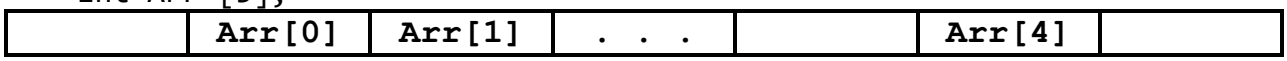
Элементы массива размещаются подряд в соответствии с ростом номера элемента внутри массива (индекса). Счет индексов начинается с нуля. Контроля выхода за границу массива нет.

2. Массивы в статической памяти

Для статических массивов память выделяется на этапе компиляции программы в стеке данных, и ее размер не может быть изменен при работе программы.

Представление статического массива в классическом C++

```
int Arr [5];
```



Все элементы располагаются на стеке данных подряд в соответствии с ростом индекса. Имя массива, это адрес его нулевого элемента:

```
Arr = &Arr[0]
```

3. Динамические массивы

Для динамических массивов память выделяется в процессе работы программы в области динамической памяти (куча – heap). Решается через механизм указателей с использованием операций new и delete. Соответственно, new выделяет память для объекта и delete разрушает объект, возвращая память в кучу (heap).

Синтаксис:

```
Указатель = new тип имя_объекта [количество];
```

Размер выделенного адресного пространства определяется в момент объявления динамического массива. Количество элементов в массиве должно быть задано константой.

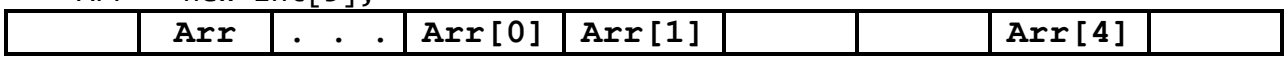
Синтаксис:

```
delete имя_объекта;
```

Для массива: `delete []имя;`

Представление динамического массива в классическом C++

```
int *Arr;
Arr = new int[5];
```



Arr размещается на стеке данных. Элементы массива размещаются в куче.
Имя массива, это синоним адреса. Arr хранит адрес массива в куче. Не изменяется при работе приложения. Высвобождается использованием операции delete:

```
delete Arr[];
```

4. Функции работы с массивами

Функции работы с массивами, как правило, получают массив через параметры. Поскольку массив, это указатель, то функция может изменять элементы массива, и это будет известно вызывающей программе.

Синтакс формального параметра, которым является массив, показан в прототипе функции:

```
float Avg(float a[], int len);
```

Или так:

```
float Avg(float *a, int len);
```

Первый параметр, это массив, второй – его фактическая длина. Лексема операции [] или знак *, присоединенный к имени массива, означает, что параметром является массив.

Семантика: масивы всегда передаются в функцию по ссылке, то есть функция будет изменять массив, если это заложено в ее алгоритме.

Заголовочный файл Pointerr.h скопируйте в папку проекта и присоедините к проекту, откройте и ознакомьтесь с содержимым. Все описания функций выполняются здесь, а управления вызовами – в Source файле проекта.

Упражнение 1. Параметры по адресу

Опишите и инициализируйте массив вещественных чисел:

```
float *Arr [] = {-1., 2.,-3.,4.,-5};
```

Найдите его емкость, она же длина:

```
int len;
len = sizeof(Arr)/sizeof(float); // Нетрудно догадаться, что len =5.
```

Передайте в функцию `transform_One`, и выведите решение функцией `Out_Arr`.

Передайте в функцию `transform_Two`, и выведите решение.

Выводы

1. Массив, это указатель. Когда в списке параметров функции массив, то функция получает адрес массива, и может изменить его значения.

2. Переменные базовых типов могут быть переданы по значению (создается копия в теле функции) или по ссылке (передается адрес объекта). Признаком ссылки является знак `&` у имени параметра. В функции `transform_Two` длина массива изменилась, поэтому она возвращается по адресу.

3. Важно понимать, что функция должна уметь защитить свои данные от несанкционированного изменения, поэтому не стоит злоупотреблять механизмами, могущими нарушить защиту.

Упражнение 2. Динамический массив. Функция, возвращающая массив

Есть задачи, когда исходный массив не должен быть изменен функцией, или функция должна породить новый массив. Например, получить копию массива.

В этом случае функция должна конструировать новый массив, и по какому-то алгоритму получить его значения, а потом вернуть. Есть два решения.

1. Новый массив возвращается через список параметров.

2. Новый массив порождается динамически, и возвращается функцией.

Описания функций получения копии массива `Copy_One` и `Copy_Two` приведены в заголовочном файле. Изучите пример, а затем получите копию массива двумя способами.

В `main` должны быть объявлены принимающие данные.

Обращение к функции `Copy_One`:

```
Copy_One(a, b, n);
```

Обращение к функции `Copy_Two`:

```
float *New_a = Copy_Two(a,n);
```

Выводы

1. Когда функция получает новые данные, она должна их вернуть в `main`. Способов вернуть данные из функции только два: через параметры или как возвращаемое значение.

2. Функция, возвращающая данное, тип которого отличен от базового, должна вернуть указатель.

3. При вызове функции, получающей новые данные, на стороне `main` должны быть объявлены объекты, принимающие значения.

Упражнение 3. Задание для самостоятельной разработки.

Опишите функцию `Two_Arr`, которая получает вещественный массив, и формирует два новых массива. В первом из них записаны только положительные элементы исходного массива, а во втором – только отрицательные. Имеет смысл возвращать массивы через список параметров. Как, собственно, и их длины.

Упражнение 4. Задание для самостоятельной разработки.

Опишите функцию `New_Arr`, которая формирует целочисленный массив указанной длины по следующему правилу: первое значение равно 1, второе = -1, третье = 2, четвертое = -2, и так далее.