**Лабораторная работа №12.**

**Указатели и ссылки. Имя массива как указатель. Динамические массивы**

## 1 Цель и порядок работы

Цель работы – изучить работу с указателями, ссылками, получить навыки программирования с использованием динамических массивов.

Порядок выполнения работы:

* ознакомиться с описанием лабораторной работы;
* получить задание у преподавателя, согласно своему варианту;
* написать программу и отладить ее на ЭВМ;
* оформить отчет.

## 2 Краткая теория

### 2.1 Указатели

Когда компилятор обрабатывает оператор определения переменной, например

int i = 10;

он выделяет память в соответствии с типом (int) и инициализирует ее указанным значением (10). Все обращения в программе к переменной по ее имени (i) заменяются компилятором на адрес области памяти, в которой хранится значение переменной.

Программист может определить собственные переменные для хранения адресов областей памяти. Такие переменные называются указателями.

Указатели предназначены для хранения адресов областей памяти. В C++ различают три вида указателей – **указатели на объект**, **на функцию** и **на void** отличающиеся свойствами и набором допустимых операций. Указатель не является самостоятельным типом, он всегда связан с каким-либо другим конкретным типом.

**Указатель на функцию** содержит адрес в сегменте кода, по которому располагается исполняемый код функции, то есть адрес, по которому передается управление при вызове функции. Указатели на функции используются для косвенного вызова функции (не через ее имя, а через обращение к переменной, хранящей ее адрес), а также для передачи имени функции в другую функцию в качестве параметра. Указатель функции имеет тип “указатель функции, возвращающей значение заданного типа и имеющей аргументы заданного типа”:

тип (\*имя) ( список типов аргументов );

Например, объявление:

int (\*fun) (double, double)

задает указатель с именем fun на функцию, возвращающую значение типа int и имеющую два аргумента типа double.

**Указатель на объект** содержит адрес области памяти, в которой хранятся данные определенного типа (основного или составного). Простейшее объявление указателя на объект (в дальнейшем называемого просто указателем) имеет вид:

тип \*имя;

где тип может быть любым, кроме ссылки и битового поля, причем тип может быть к этому моменту только объявлен, но еще не определен (следовательно, в структуре, например, может присутствовать указатель на структуру того же типа).

Звездочка относится непосредственно к имени, поэтому для того, чтобы объявить несколько указателей, требуется ставить ее перед именем каждого из них. Например, в операторе

int \*a, b, \*c;

описываются два указателя на целое с именами **а** и **с**, а также целая переменная **b**. Размер указателя зависит от модели памяти. Можно определить указатель на указатель и т.д.

**Указатель на void** применяется в тех случаях, когда конкретный тип объекта, адрес которого требуется хранить, не определен (например, если в одной и той же переменной в разные моменты времени требуется хранить адреса объектов различных типов).

Указателю на **void** можно присвоить значение указателя любого типа, а также сравнивать его с любыми указателями, но перед выполнением каких-либо действий с областью памяти, на которую он ссылается, требуется преобразовать его к конкретному типу явным образом.

Существует специальная операция взятия адреса, обозначаемая символом &. Ее результатом является адрес объекта.

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

int j; // целая переменная

const int ci = 1; // целая константа

int \*рi; // указатель на целую переменную

const int \*pci; // указатель на целую константу

int \* const ср = &i; // указатель-константа на целую переменную

const int \* const срс = &ci // указатель-константа на целую константу

Указатели чаще всего используют при работе с динамической памятью или кучей (от англ. heap). Это свободная память, в которой можно во время выполнения программы выделять место в соответствии с потребностями. Доступ к выделенным участкам динамической памяти, называемым динамическими переменными, производится только через указатели. Время жизни динамических переменных – от точки создания до конца программы или до явного освобождения памяти. В C++ используется два способа работы с динамической памятью. Первый использует семейство функций malloc и достался в наследство от языка C, второй использует операции new и delete.

При определении указателя надо стремиться выполнить его инициализацию, то есть присвоение начального значения. Непреднамеренное использование неинициализированных указателей – распространенный источник ошибок в программах. Инициализатор записывается после имени указателя либо в круглых скобках, либо после знака равенства.

Существуют следующие способы инициализации указателя:

1. Присваивание указателю адреса существующего объекта:

• с помощью операции получения адреса:

int а = 5; // целая переменная

int\* p = &a; // в указатель записывается адрес а

int\* р (&а); // то же самое другим способом

• с помощью значения другого инициализированного указателя:

int\* r = р;

• с помощью имени массива или функции, которые трактуются как адрес:

int b[10]; // массив

int\* t = b; // присваивание адреса начала массива

void f(int а){ /\* ... \*/ } // определение функции

void (\*pf) (int); // указатель на функцию

pf = f; // присваивание адреса функции

2. Присваивание указателю адреса области памяти в явном виде:

char\* cp = (char \*) 0хВ8000000;

Здесь 0хВ8000000 — шестнадцатеричная константа, **(char** **\*)** – операция приведения типа: константа преобразуется к типу «указатель на char».

3. Присваивание пустого значения:

int\* suxx = NULL;

int\* rulez = 0;

В первой строке используется константа NULL, определенная в некоторых заголовочных файлах C как указатель, равный нулю. Рекомендуется использовать просто 0, так как это значение типа int будет правильно преобразовано стандартными способами в соответствии с контекстом.

4. Выделение участка динамической памяти и присваивание ее адреса указателю:

• с помощью операции new:

int\* n = new int; // 1

int\* m = new int (10); // 2

int\* q = new int [10]; // 3

• с помощью функции mallос:

int\* u = (int \*)malloc(sizeof(int)); // 4

В операторе 1 операция new выполняет выделение достаточного для размещения величины типа int участка динамической памяти и записывает адрес начала этого участка в переменную n. Память под саму переменную n (размера, достаточного для размещения указателя) выделяется на этапе компиляции. В операторе 2, кроме описанных выше действий, производится инициализация выделенной динамической памяти значением 10.

В операторе 3 операция new выполняет выделение памяти под 10 величин типа int (массива из 10 элементов) и записывает адрес начала этого участка в переменную q, которая может трактоваться как имя массива. Через имя можно обращаться к любому элементу массива.

В операторе 4 делается то же самое, что и в операторе 1, но с помощью функции выделения памяти malloc, унаследованной из библиотеки С. В функцию передается один параметр – количество выделяемой памяти в байтах.

Операцию new использовать предпочтительнее, чем функцию malloc, особенно при работе с объектами.

Освобождение памяти, выделенной с помощью операции new, должно выполняться с помощью delete, а для выделенной с помощь malloc осовобождаться с помощью free.

Для описанных выше переменных освобождение памяти будет выглядеть так:

delete n; // 1

delete m; // 2

delete []q; // 3

free(u); // 4

При выделении памяти с помощью new[], необходимо применять delete[]. Размерность массива при этом не указывается.

### 2.2 Операции с указателями

С указателями можно выполнять следующие операции: разыменовывание (или косвенное обращение к объекту) (\*), присваивание, сложение с константой, вычитание, инкремент (++), декремент ( -- ), сравнение, приведение типов.

При работе с указателями часто используется операция получения адреса &.

**Операция разыменования** предназначена для доступа к величине, адрес которой хранится в указателе. Эту операцию можно использовать как для получения, так и для изменения значения величины (если она не объявлена как константа):

char а; // переменная типа char

char \*р = new char; /\* выделение памяти под указатель (char \*р) и

под динамическую переменную типа char (new char)\*/

\*р = 'Ю'; // присваивание значения динамической переменной

// копирование содержимого ячейки памяти на которую указывает p в переменную a

а = \*р;

Как видно из примера, конструкцию \*имя указателя можно использовать в левой части оператора присваивания. С ней допустимы все действия, определенные для величин соответствующего типа (если указатель инициализирован). На одну и ту же область память может ссылаться несколько указателей различного типа.

**Арифметические операции с указателями** (сложение с константой, вычитание, инкремент и декремент) автоматически учитывают размер типа величин, адресуемых указателями. Эти операции применимы только к указателям одного типа и имеют смысл в основном при работе со структурами данных, последовательно размещенными в памяти, например, с массивами.

**Инкремент** перемещает указатель к следующему элементу массива, **декремент** – к предыдущему. Фактически значение указателя изменяется на величину sizeof**[[1]](#footnote-1)**(тип). Если указатель на определенный тип увеличивается или уменьшается на константу, его значение изменяется на величину этой константы, умноженную на размер объекта данного типа, например:

short \*р = new short [5];

p++; // значение р увеличивается на 2 байта

long \*q = new long [5];

q++; // значение q увеличивается на 4 байта

**Разность двух указателей** – это разность их значений, деленная на размер типа в байтах (в применении к массивам разность указателей, например, на третий и шестой элементы равна 3). Суммирование двух указателей не допускается.

При записи выражений с указателями следует обращать внимание на приоритеты операций. В качестве примера рассмотрим последовательность действий, заданную в операторе

Унарная **операция получения адреса** & применима к величинам, имеющим имя и размещенным в оперативной памяти. Таким образом, нельзя получить адрес скалярного выражения, неименованной константы или регистровой переменной.

int а = 5; // переменная a

int \*p = &a; // в указатель записывается адрес а

### 2.3 Ссылки

Ссылка представляет собой синоним имени, указанного при инициализации ссылки. Ссылку можно рассматривать как указатель, который всегда разыменовывается. Формат объявления ссылки:

тип & имя;

где тип — это тип величины, на которую указывает ссылка, & — оператор ссылки, означающий, что следующее за ним имя является именем переменной ссылочного типа, например:

int kol;

int& раl = kol: // ссылка раl — альтернативное имя для kol

Переменная-ссылка должна явно инициализироваться при ее описании, кроме случаев, когда она является параметром функции, описана как extern или ссылается на поле данных класса

После инициализации ссылке не может быть присвоена другая переменная.

Тип ссылки должен совпадать с типом величины, на которую она ссылается.

Не разрешается определять указатели на ссылки, создавать массивы ссылок и ссылки на ссылки.

Ссылки применяются чаще всего в качестве параметров функций и типов возвращаемых функциями значений. Ссылки позволяют использовать в функциях переменные, передаваемые по адресу, без операции разыменовывания, что улучшает читаемость программы.

Ссылка, в отличие от указателя, не занимает дополнительного пространства в памяти и является просто другим именем величины. Операция над ссылкой приводит к изменению величины, на которую она ссылается.

### 2.4 Указатели и массивы

Доступ к произвольному элементу массива обеспечивается по имени массива и индексу – целочисленному смещение от начала:

имя\_массива [ индекс ]

**Имя массива является указателем-константой**. К нему приемлемы все правила адресной арифметики, связанные с указателями. Для любого массива соблюдается равенство:

имя\_массива == &имя\_массива == &имя\_массива[ ]

Доступ к элементам массива возможен по его имени как указателю. Если описан массив

int z[3];

то \*zаналогично z**[**0**],** а **\*(**z**+**1**)** **-** z**[**1**]** и т.д. Т.е.

z**[**0**]** == **\*(**z**)** == **\*(**z **+** 0**)**

z**[**1**]** == **\*(**z **+** 1**)**

z**[**2**]** == **\*(**z **+** 2**)**

z**[**i**]** == **\*(**z **+** i**)**

Так как имя массива есть не просто указатель, а указатель константа, то значение имени невозможно изменить. Получить доступ ко второму элементу массива **int** x**[**4**]** c помощью выражения **\*(++**z**)** будет ошибкой, а выражение **\*(**x **+** 1**)** допустимо.

Рассмотрим такой пример обращения к массиву: вывести на экран элементы массива, используя имя массива как указатель.

#include <iostream>

using namespace std;

void main()

{

 int x[ ]= {1, 2, 3, 4, 5, 0};

 int i = 0;

 int size = sizeof(x)/sizeof(x[0]);

 while (i < size)

 {

 cout << endl << \*(x + i); //аналогично x[i]

 i++;

 }

}

*Замечание*.

Обращение к элементу массива относят к постфиксному выражению вида PE[IE]. PE – указатель на нужный тип, IE – целочисленный тип, PE[IE] – индексированный элемент этого массива. Аналогично при использовании адресной арифметики \*(PE + IE). Поскольку сложение коммутативно, то запись \*(PE + IE) эквивалентна \*(IE + PE), а, следовательно, и IE[PE] именует тот же элемент массива, что и PE[IE]. Т.е. приведенного выше примера будет справедливо:

\*(x + i) == \*(i + x) == i[x] == x[i]

### 2.5 Динамические массивы

Если до начала работы программы неизвестно, сколько в массиве элементов, в программе следует использовать динамические массивы. Память под них выделяется с помощью операции new или функции malloc в динамической области памяти во время выполнения программы. Адрес начала массива хранится в переменной, называемой указателем. Например:

int n = 10:

int \*а = new int[n];

Во второй строке описан указатель на целую величину, которому присваивается адрес начала непрерывной области динамической памяти, выделенной с помощью операции new. Выделяется столько памяти, сколько необходимо для хранения n величин типа int. Величина n может быть переменной.

Обнуления памяти при ее выделении не происходит. Инициализировать динамический массив нельзя.

Обращение к элементу динамического массива осуществляется так же, как и к элементу обычного – например а[3]. Можно обратиться к элементу массива и другим способом – \*(а + 3) . В этом случае мы явно задаем те же действия, что выполняются при обращении к элементу массива обычным образом. Рассмотрим их подробнее. В переменной-указателе а хранится адрес начала массива. Для получения адреса третьего элемента к этому адресу прибавляется смещение 3. Операция сложения с константой для указателей учитывает размер адресуемых элементов, то есть на самом деле индекс умножается на длину элемента массива: а + 3 \* sizeof(int). Затем с помощью операции \*(разыменовывание) выполняется выборка значения из указанной области памяти.

Если динамический массив в какой-то момент работы программы перестает быть нужным и мы собираемся впоследствии использовать эту память повторно, необходимо освободить ее с помощью операции delete[]. Квадратные скобки в операции delete [] при освобождении памяти из-под массива обязательны. Их отсутствие может привести к неопределенному поведению программы.

Таким образом, время жизни динамического массива, как и любой динамической переменной, с момента выделения памяти до момента ее освобождения. Область действия зависит от места описания указателя, через который производится работа с массивом. Локальная переменная при выходе из блока, в котором она описана, «теряется». Если эта переменная является указателем и в ней хранится адрес выделенной динамической памяти, при выходе из блока эта память перестает быть доступной, однако не помечается как свободная, поэтому не может быть использована в дальнейшем. Это называется **утечкой памяти** и является распространенной ошибкой.

### 2.6 Примеры программ

Пример 1. Программа, демонстрирующая работу с указателями и ссылками.

#include "stdafx.h"

#include <iostream>

#include <stdio.h>

using namespace std;

void main()

{

 setlocale(LC\_ALL, "Russian");

 int a = 1, b;

 int \*x, \*y = &b; //

 b = 3;

 x = &a;

 int \*z = new int(5);

 int &k = b;

 printf("+-------+---+---+--------+--------+---+---+---+--------+--------+---+--------+\n");

 printf("| cmd | a |\*x | &a | x | b |k |\*y | &b | y |\*z | z |\n");

 printf("+-------+---+---+--------+--------+---+---+---+--------+--------+---+--------+\n");

 printf("| | %1d | %1d | %6X | %6X | %1d | %1d | %1d | %6X | %6X | %1d | %6X |\n", a, \*x, &a, x, b, k, \*y, &b, y, \*z, z);

 //увеличим значение ячейки памяти на которую указывает x, (т.е. переменную b), при этом поменяется и b

 (\*x)++;

 printf("|(\*x)++ | %1d | %1d | %6X | %6X | %1d | %1d | %1d | %6X | %6X | %1d | %6X |\n", a, \*x, &a, x, b, k, \*y, &b, y, \*z, z);

 //присвоим b значение -1, при этом поменяется и \*x

 b = 7;

 printf("|b = 7 | %1d | %1d | %6X | %6X | %1d | %1d | %1d | %6X | %6X | %1d | %6X |\n", a, \*x, &a, x, b, k, \*y, &b, y, \*z, z);

 //скопируем значение ячейки памяти на которую указывает z в ячейку на которую указывает y

 \*y = \*z;

 printf("|\*y = \*z| %1d | %1d | %6X | %6X | %1d | %1d | %1d | %6X | %6X | %1d | %6X |\n", a, \*x, &a, x, b, k, \*y, &b, y, \*z, z);

 //тепеь x указывает туда же, куда и y, т.е. на b, значение \*x поменялось

 x = y;

 printf("|x = y | %1d | %1d | %6X | %6X | %1d | %1d | %1d | %6X | %6X | %1d | %6X |\n", a, \*x, &a, x, b, k, \*y, &b, y, \*z, z);

 //изменим значение a, другие значения не изменились, так как x на a больше никто не указывает

 a \*= 3;

 printf("|a \*= 3 | %1d | %1d | %6X | %6X | %1d | %1d | %1d | %6X | %6X | %1d | %6X |\n", a, \*x, &a, x, b, k, \*y, &b, y, \*z, z);

 //поменялись значения указателей, которые указывали на b и сама b

 \*x = 9;

 printf("|\*x = 9 | %1d | %1d | %6X | %6X | %1d | %1d | %1d | %6X | %6X | %1d | %6X |\n", a, \*x, &a, x, b, k, \*y, &b, y, \*z, z);

 //теперь x указывает туда же, куда и z, т.е. на переменную в динамической памяти,

 //значение \*y поменялось и адрес y тоже

 y = z;

 printf("|y = z | %1d | %1d | %6X | %6X | %1d | %1d | %1d | %6X | %6X | %1d | %6X |\n", a, \*x, &a, x, b, k, \*y, &b, y, \*z, z);

 //изменим значение b через ссылку k

 k = 7;

 printf("|k = 7 | %1d | %1d | %6X | %6X | %1d | %1d | %1d | %6X | %6X | %1d | %6X |\n", a, \*x, &a, x, b, k, \*y, &b, y, \*z, z);

 //изменим значение b через ссылку k

 k -= 3;

 printf("|k -= 3 | %1d | %1d | %6X | %6X | %1d | %1d | %1d | %6X | %6X | %1d | %6X |\n", a, \*x, &a, x, b, k, \*y, &b, y, \*z, z);

 //изменим значение z

 \*z += 3;

 printf("|\*z += 3| %1d | %1d | %6X | %6X | %1d | %1d | %1d | %6X | %6X | %1d | %6X |\n", a, \*x, &a, x, b, k, \*y, &b, y, \*z, z);

printf("+-------+---+---+--------+--------+---+---+---+--------+--------+---+--------+\n");

 delete z;

}

Результаты работы программы:

+-------+---+---+--------+--------+---+---+---+--------+--------+---+--------+

| cmd | a |\*x | &a | x | b |k |\*y | &b | y |\*z | z |

+-------+---+---+--------+--------+---+---+---+--------+--------+---+--------+

| | 1 | 1 | 12FF60 | 12FF60 | 3 | 3 | 3 | 12FF54 | 12FF54 | 5 | 3159C0 |

|(\*x)++ | 2 | 2 | 12FF60 | 12FF60 | 3 | 3 | 3 | 12FF54 | 12FF54 | 5 | 3159C0 |

|b = 7 | 2 | 2 | 12FF60 | 12FF60 | 7 | 7 | 7 | 12FF54 | 12FF54 | 5 | 3159C0 |

|\*y = \*z| 2 | 2 | 12FF60 | 12FF60 | 5 | 5 | 5 | 12FF54 | 12FF54 | 5 | 3159C0 |

|x = y | 2 | 5 | 12FF60 | 12FF54 | 5 | 5 | 5 | 12FF54 | 12FF54 | 5 | 3159C0 |

|a \*= 3 | 6 | 5 | 12FF60 | 12FF54 | 5 | 5 | 5 | 12FF54 | 12FF54 | 5 | 3159C0 |

|\*x = 9 | 6 | 9 | 12FF60 | 12FF54 | 9 | 9 | 9 | 12FF54 | 12FF54 | 5 | 3159C0 |

|y = z | 6 | 9 | 12FF60 | 12FF54 | 9 | 9 | 5 | 12FF54 | 3159C0 | 5 | 3159C0 |

|k = 7 | 6 | 7 | 12FF60 | 12FF54 | 7 | 7 | 5 | 12FF54 | 3159C0 | 5 | 3159C0 |

|k -= 3 | 6 | 4 | 12FF60 | 12FF54 | 4 | 4 | 5 | 12FF54 | 3159C0 | 5 | 3159C0 |

|\*z += 3| 6 | 4 | 12FF60 | 12FF54 | 4 | 4 | 8 | 12FF54 | 3159C0 | 8 | 3159C0 |

+-------+---+---+--------+--------+---+---+---+--------+--------+---+--------+

Пример 2. Найти сумму элементов в динамическом массиве. Размерность массива ввести с клавиатуры. Использовать различные варианты при обращении к массиву.

#include "stdafx.h"

#include <iostream>

using namespace std;

void main()

{

 setlocale(LC\_ALL, "Russian");

 //введем размерность массива

 int N;

 cout << "Введите размерность массива: ";

 cin >> N;

 //создаем массив динамически

 int \*a = new int[N];

 //введем элементы массива с клавиатуры

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

 {

 cout << "Введите " << i <<"-й элемент массива: ";

 cin >> a[i];

 }

 //объявим переменную для хранения суммы элементов

 int s = 0;

 //просуммируем элементы массива

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

 s += \*(a + i);

 //выведем на экран элементы массива и их сумму

 cout << "a:";

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

 cout << " " << i[a];

 cout << endl << "Сумма элементов: " << s << endl;

 //очищаем выделенную память

 delete[] a;

}

##  Варианты заданий

### 5.1 Указатели

Используя вместо самой переменной указатель на нее, написать программу в соответствии с заданием:

* 1. Вычислить площадь треугольника по стороне и высоте.
	2. Вычислить площадь окружности по заданному радиусу.
	3. Даны значения a и b, найти их среднее арифметическое, среднегеометрическое.
	4. Вычислить y = tg(x)+5x3-4x2.
	5. Вычислить площадь квадрата.
	6. Вычислить высоту треугольника, зная две стороны треугольника и угол между ними.
	7. Вычислить y = |x-cos(x)|.
	8. Ввести сторону квадрата a. Вычислить площадь вписанной окружности.
	9. Задается длина окружности. Найти площадь круга, ограниченного этой окружностью.
	10. Вычислить углы треугольника, зная его стороны.
	11. Вычислить площадь трапеции.
	12. Вычислить y = cos|x3-x2|.
	13. Вычислить длину гипотенузы прямоугольного треугольника, зная длины двух катетов.
	14. Вычислить корень квадратный от (x5-x4+|x3|).
	15. Вычислить корень квадратный от (sin(x)+cos(x)).
	16. Вычислить объем цилиндра, зная радиус основания и высоту.
	17. Вычислить объем конуса.
	18. Определить время, через которое встретятся два тела, равноускоренно движущиеся друг к другу. Известны: v1 и v2 - начальные скорости, a1 и a2 - ускорения, s - расстояние между ними.
	19. Вычислить сторону треугольника, зная две другие стороны и угол между ними.
	20. Вычислить площадь ромба, зная длину стороны и угол.
	21. Вычислить площадь треугольника, зная длины всех сторон и радиус описанной окружности.
	22. Вычислить у = sin(x)\*cos(x)-3x2.
	23. Определить время падения камня на поверхность Земли с высоты h.
	24. Вычислить y = |x-tg(x)|.
	25. Вычислить y = tg(x3)+|x2-x5|.
	26. Вычислить высоту равностороннего треугольника, зная длину всех сторон.
	27. Вычислить y = x2-sin(x)+cos(x).
	28. Вычислить y = x3-x5+|x-sin(x)|+x2.
	29. Вычислить гипотенузу треугольника, зная длины двух катетов.
	30. Найти корни квадратного уравнения Аx2 - Вх + С=0 для А, В и С, вводимых с клавиатуры.
	31. Вычислить длину отрезка, зная координаты его концов.
	32. Для А, вводимого с клавиатуры вычислить В=А^10 за четыре операции.
	33. Вычислить y = x/cos (x) + x2/sin (x).
	34. Вычислить среднее арифметическое четырех вводимых чисел.
	35. Вычислить среднее геометрическое пяти вводимых чисел.

### 5.2 Ссылки

Написать программу, в соответствии с заданием из пункта 5.1, но вместо самой переменной использовать ссылку на нее.

1. Оператор **sizeof** возвращает размер в байтах объекта или типа данных. Синтаксис его таков:

sizeof ( type name );

sizeof ( object );

sizeof object; [↑](#footnote-ref-1)