Лабораторная работа № 10

«**Разработка, отладка и испытание алгоритмов и программ с использованием ссылок и указателей**»

Для каждого учащегося предназначено 3 задания (№ 0, 1 и 2). Задание 2 состоит из пяти подзаданий, так что каждому учащемуся нужно выполнить всего **7 заданий**, соответствующих его номеру по списку группы.

Задания можно выполнить либо в отдельных проектах одного решения, либо в одном проекте – одной программе с консольным меню

Использовать **косвенную адресацию** элементов динамического массива,
для одномерного по типу « ***(array + i)** », для двумерного **« *(*(massiv + i) + j)** »

Задание 0.

Прочитать и разобраться в предложенном материале и примерах:

На защите данной лабораторной работы учащийся должен уметь создавать указатели, ссылки на переменные; распечатывать на консоли адреса переменных и указателей.

Указатель - это переменная или константа, содержащая адрес другой переменной.
Значение указателя - это беззнаковое целое число длиной 4 байта. Оно сообщает, где расположена переменная и ничего не говорит о самой переменной. Тип указателя сообщает нам тип переменной, на которую он ссылается, при условии, что это не «универсальный» указатель типа void.
В некоторых случаях без указателей не обойтись, и часто программы с ними короче и эффективнее.
Как и всякая переменная, указатель должен быть объявлен. При объявлении указателя всегда указывается тип переменной, на которую он может указывать, перед именем указателя ставится символ *. Часто в имени указателя используют букву p (pointer).

```
float a, *pa;// переменная float, указатель на float
int *pi, *pj;// два указателя на int
char c, *pc;// переменная char, указатель на char
```

Указатель инициализируется адресом переменной соответствующего типа. Для взятия адреса используется унарная операция &. Справа от & может стоять только адресное выражение.

```
pa = &a;
int i = 4;
pj = &i;
c = '+';
char *p1 = &c;
```

Для получения значения переменной с помощью указателя используется унарная операция * (разыменование):

```
cout << *p1 << *pj;// будет выведено +4
cout << p1 << pj;// будут выведены 2 адреса
int k = 7, h = 8;
int *pk = &k, *ph = &h;
*pk *= *ph;// в переменной k число 56, аналогично k *= 8 (k = k*h = 7*8)
```

Указатели удобно использовать при работе с одномерными и многомерными массивами. В случае одномерного массива указателем на массив является имя самого массива или указатель на его нулевой элемент:

```
int m[10] = {0,1,2,3,4,5,6,7,8,9};
int *p1, *p2;
p1 = m;
p2 = &m[0];// в переменных p1 и p2 один и тот же адрес
cout << m + 3 << " " << &m[3] << endl;// один и тот же адрес (больше предыдущего)
cout << m[3] << " " << *(p1+3) << endl;// один и тот же элемент (3)
```

В случае многомерного массива следует помнить, что это - массив массивов:

```
int x[][2]= {1,2,3,4,5,6,7,8,9,10,11,12};
int *px1 = &x[0][0];// адрес начала массива
int *px2 = x[0];// тоже самое
cout << px1 << " " << px2;// один и тот же адрес
```

Операции с указателями.

Название	Знак	Пояснение
Взятие адреса	&	Получить адрес переменной
Разыменование	*	Получить значение переменной по адресу
Присваивание	=	Присвоить указателю адрес переменной или 0
Инкремент	++	Увеличить указатель на целое значение (на след. элемент массива)
Декремент	--	Уменьшить указатель на целое значение (на пред. элемент массива)

Сложение	+	Увеличить указатель на целое значение и присвоить другому указателю
Сложение с замещением	+=	Увеличить существующий указатель на целое значение
Вычитание	–	Уменьшить указатель на целое значение или на значение другого указателя, если оба указывают на один и тот же массив, и присвоить третьему указателю.
Вычитание с замещением	–=	Уменьшить указатель на целое значение или на значение другого указателя, если оба указывают на один и тот же массив.
Отношения	== != < <= > >=	Сравнение указателей – истина или ложь

Арифметические операции с указателями позволяют смещаться по элементам массива. Обратите внимание, что для двумерного массива уже не требуются два цикла:

```
int x[][2]= {1,2,3,4,5,6,7,8,9,10,11,12};
int *px = x[0];
// вывод всех элементов массива
for (int i = 0; i < 12; i++)
    cout << *(px+i) << " ";
// то же самое, но теперь указатель px указывает за пределы массива
while (px < x[0]+12)
    cout << *px++ << " ";
// так можно убедиться, что каждый раз указатель смещается на 4 байта, т.е. на длину элемента массива (int)
px = x[0];
while (px < x[0]+12)
    cout << px++ << " ";
// px указывал на конец массива, так можно определить размер массива (количество элементов)
cout << px - x[0] << endl;
```

Строки интерпретируются как обычные массивы с размером каждого элемента в 1 символ (1 байт). Имя строки – это адрес.

Неразыменованный указатель на char рассматривается как строка от текущей позиции до символа '\0'.
Разыменованный указатель на char - это просто отдельный символ.

```
char s[] = "Privet, Alice!";
char *p = s;
while(*p)//или *p != NULL
{
    cout << *(p++);//вывод отдельного символа (разыменованный указатель)
    cout << p++;//вывод до конца строки (неразыменованный указатель)
}
```

Для обычных переменных память выделяется автоматически при запуске программы, в которой они объявлены, и также автоматически освобождается при завершении программы. Доступ к переменной происходит по ее идентификатору.

Каждая ячейка оперативной памяти имеет свой номер или адрес (0, 1, 2, …, 00ff2aa0, …,).

Наименьшая адресуемая единица оперативной памяти – 1 байт, а на диске (съемном устройстве постоянного хранения данных) – кластер.

Язык С++ позволяет обратиться к значению переменной не только по ее имени, но и при помощи ее адреса, по которому записано (хранится) ее значение.

Указатель (pointer) - это переменная, содержащая адрес другой переменной (объекта). Точнее - адрес первого байта области оперативной памяти, выделенной под хранение этого объекта (переменной). Это дает возможность косвенного доступа к этому объекту через указатель.

Ячейка с переменной X	Ячейка с указателем
5	0012FF70
Адрес переменной X	Адрес указателя
0012FF70	0012FF6C

В 32-разрядных приложениях адрес занимает 4 байта. Следовательно, любой указатель имеет длину 4 байта. Принято соглашение, по которому имя переменной-указателя должно начинаться с буквы p.

Объявление указателя.

Указатель, как любая переменная должен быть объявлен в программе до его использования.
int* pa;// pa – это переменная, хранящая адрес ячейки с целым значением int; тип этой переменной – указатель на int (int*)
int *pa, *pb, *pc; // объявление нескольких указателей одного типа int
Тип указателя должен совпадать с типом переменной, адрес которой он хранит.
Контроль за типом указателя в C++ более строгий, чем за типом переменной:
int a = 5;
float b = 3.4;
a = b; // это возможно
int *pa;
float *pb;
pa = pb; // ошибка
Инициализация указателя.
Нельзя использовать в программе указатель, значение которого не определено (но ошибки это не вызовет).
Можно проинициализировать при объявлении:
double* px = 0; // это указатель в никуда
double* px = NULL; // тоже самое, но более грамотно. NULL – это указатель void'овского (пустого) типа на ноль (void*)0
double y;
double *py = &y; // взять адрес переменной y и поместить его (адрес переменной y) в указатель py
Операция &
Унарная операция & (взятие или получение адреса) выдает адрес объекта, так что оператор
px = &x;
присваивает переменной px адрес переменной x. Говорят, что px "указывает" на x.
Не путайте унарный оператор & («взять адрес», например, &x) с бинарным оператором & («логическое И», например, x<45 & y==20). Они различаются по контексту («логическое И» обычно будет в проверках if, if-else, сравнениях, причем слева и справа от & будут выражения; в то время как «взятие адреса» будет только с операндом справа, например &y).
Операция & применима только к адресным выражениям (ℓ-выражения, L-value, left-value), так что конструкции вида &(x-1) и &3 незаконны (нет отдельной ячейки памяти, хранящей значение x-1 или 3).
Операция *
Унарная операция * называется операцией разыменования (разадресации, операцией разрешения адреса). Эта операция извлекает значение по указанному адресу.
double x = 3.2, y = 5.7, z;
double *px = &x, *py = &y;
z = *px + *py; // значение по адресу px + значение по адресу py = 8.9
Само объявление указателя теперь обретает смысл:
double *px – разыменование px дает double.
Операция =
Оператор присвоения = помещает (как и везде) значение, записанное справа от себя в переменную (указатель, ссылку, константу), записанную слева от себя.
px = &x;//адрес икса поместить в указатель py (py должен быть создан как указатель, а не переменная)
py = NULL; // указатель в никуда; «пустой адрес, ноль-адрес» поместить в указатель py, то есть «занулить» указатель, сделать его указателем на никакой адрес. Разумеется, потом указателю py можно присвоить «настоящий» адрес какой-либо переменной, подходящей по типу (например, и там, и там должен быть тип int или другой, но обязательно совпадающий между типом переменной и типом указателя на нее).
Операции с указателями.

Название	Знак	Пояснение
Взятие адреса	&	Получить адрес переменной
Разыменование	*	Получить значение переменной по адресу
Присваивание	=	Присвоить указателю адрес переменной или NULL
Инкремент	++	Увеличить указатель на целое значение (указатель станет указывать на следующий элемент массива)
Декремент	--	Уменьшить указатель на целое значение (указатель станет указывать на предыдующий элемент массива)
Сложение	+	Увеличить указатель на целое значение и присвоить другому указателю
Сложение с замещением	+=	Увеличить существующий указатель на целое значение

Вычитание	–	Уменьшить указатель на целое значение или на значение другого указателя, если оба указывают на один и тот же массив, и присвоить третьему указателю.
Вычитание с замещением	–	Уменьшить указатель на целое значение или на значение другого указателя, если оба указывают на один и тот же массив.
Отношения	== != < <= > >=	Сравнение указателей, то есть адресов (это числа), которые они хранят. В результате можно получить значение – истина или ложь, означающая соответственно, совпадение, несовпадение адресов, или расположение адресов левее или правее
Выделение памяти	new	Получить указатель на начало выделенного блока памяти
Освобождение памяти	delete	Освободить выделенный блок памяти и сделать указатель недоступным

Указатели и массивы.

Имя одномерного массива само по себе является указателем.

```
int x[] = {1,2,3,4,5,6,7,8,9,10,11,12};
cout << x; // выведет какой-то адрес, например 0x0012FF50
int* px1 = &x[0]; // берем адрес первого элемента массива x
int* px2 = x; // не требуется использовать &, ведь x – это указатель, то есть он на самом деле int* x и хранит в себе адрес начала статического массива
```

Но это работает только с одномерными массивами.

```
int y[][2]= {1,2,3,4,5,6,7,8,9,10,11,12};
int* py = y; // так делать нельзя, ведь *py – указатель «первого уровня», а int** y – указатель «второго уровня», поскольку он указатель на строки и столбцы
```

А вот так можно:

```
int* py1 = &x[0][0];
int* py2 = x[0];
```

А это уже другой адрес:

```
int y[][2]= {1,2,3,4,5,6,7,8,9,10,11,12};
int* py3 = y[1];
cout << *py3; // *py3 вернет – нулевой элемент первой строки, то есть y[1][0] – число 3.
```

Указатель можно складывать с целым.

Если к указателю pa прибавляется целое приращение i, то приращение масштабируется размером памяти, занимаемой объектом, на который указывает указатель pa.

Таким образом, pa+i - это адрес i-го элемента после pa, причем считается, что размер всех этих i элементов равен размеру объекта, на который указывает pa. Это имеет смысл только с массивами.

```
int x[]= {1,2,3,4,5,6,7,8,9,10,11,12};
cout << x + 5 << &x[5]; // один и тот же адрес
cout << x[5] << *(x+5); // увидим 6
int a = 6, b = 7, c = 4, *pa = &a, *pb = &b;
pa +=2;
cout << "pa = " << pa << endl; // это адрес на 2*4 больше
cout << "a = " << *pa << endl; // здесь мы увидим ерунду (мусорные данные, которые здесь оказались от работы предыдущего кода программ)
```

Итак, если a - массив, то

a+i - адрес i-го элемента этого массива, т.е.

&a[i] равен a+i и a[i] равняется *(a+i).

Операции ++ и – –

```
int *px, *py;
int x[3], y[5];
px = x; //0x0012FF74
px++; // следующий элемент x 0x0012FF78
py = &y[4]; // 0x0012FF8C
py--; // предыдущий элемент y 0x0012FF88
```

Все унарные операции имеют 2-ой приоритет (выше только скобки).

Приоритет	Операция	Примечание	Порядок выполнения
1	:: -> . [ ] ( ) (int)	разрешение контекста, извлечение индексирование массива вызов функции преобразование типа	слева - направо слева - направо слева - направо слева - направо
2	++ -- ~ ! - + & * new, delete sizeof	 унарный -, унарный + получение адреса разрешение указателя работа с динамической памятью определение размера	справа - налево справа - налево справа - налево справа - налево справа - налево справа - налево

(*px)++; // x = x + 1
*px ++; // нарастит px, а затем разыменует полученный указатель
++ *px; // x = x + 1
+= –= увеличит или уменьшит на целое число.
px += 5;
py -= 4; // учтите, что выход за пределы массива не контролируется
Имя массива – константа (константный указатель на начало массива (на нулевой элемент массива))
int x[3]={0};
x++; // так делать нельзя
x=x+2; // и так тоже нельзя

Указатели можно сравнивать.

Если p и q указывают на элементы одного и того же массива, то такие отношения, как < >= и т.д. работают надлежащим образом. Например, p < q истинно, т.е. == 1, если p указывает на более ранний (находящийся левее) элемент массива, чем q. Любой указатель можно сравнить на равенство или неравенство с так называемым нулевым указателем NULL, который ни на что не указывает. Однако не рекомендуется сравнивать указатели, указывающие на различные массивы.

px = &x[9];
py = &x[4];
px > py

Указатели можно вычитать.

Если p и q указывают на элементы одного и того же массива, то p - q дает количество элементов массива между p и q.

px – py;// даст 5
py – px;// даст - 5

Работа с многомерными массивами.

```
int x[][3][2]= {1,2,3,4,5,6,7,8,9,10,11,12};
int *px = &x[0][0][0];
for (int i =0; i < 12; i++) // вывод всех элементов массива x
{
	cout << *(px++) << "\t";
}
```

Работа со строками.

Строки интерпретируются как обычные массивы с размером каждого элемента в 1 символ (1 байт). Имя строки – это адрес.

Вывод обычного массива:

```
int s[10] = {1,2,3,4,5,6,7,8,9,10};
int *p = s;
int i = 10;
while (i--)//пока i != 0
{
	cout << *(p++) << endl;
}
//на консоль напечатает:
1
2
3
4
5
```

6
7
8
9
10
Вывод такого же строкового массива:
```
char s[10] = "irina";
char *p = s;
while (*p)//пока p != NULL
{
	cout << *(p++) << endl;
}
```
//на консоль напечатает:
i
r
i
n
a
Большинство команд умеют работать со строками: они интерпретируют указатель на массив char как начало строки и обрабатывают строку до нуль-символа (‘\0’, «ноль-терминатора»).
Но если убрать разыменование:
```
char s[10] = "irina";
char *p = s;
while (*p)//пока p != NULL
{
	cout << p++ << endl;
}
```
//на консоль напечатает:
irina
rina
ina
na
a

Использование указателей.

Указатели на указатели.

В языке C++ есть возможность описать переменные-указатели, указывающие на другие указатели.

ppi	pi	i
0x00123030	0x00122020	0x00121010

```
int i = 10;
int* pi = &i;
int** ppi = &pi;
int*** pppi = &ppi;
```
Но нельзя написать:
```
&&i
&(&i)
```
Взять адрес можно только у переменной, а &i не является переменной.

Указатели на void.

Если заранее неизвестно, на какой именно тип данных будет ссылаться указатель, его описывают как void. Это не совсем тоже самое, что тип возвращаемого функцией значения.
void:
```
void main(){} // функция ничего не возвращает (пусто)
```
Указатель на void ссылается на любой тип данных:
```
void *pv;
int a;
float c;
double d;
pv = &a;
pv = &c;
pv = &d;
int *pa = pv; //ошибка cannot convert from 'void *' to 'int *'
```

С таким указателем не будут работать арифметические операции:
++ pobject; // ошибка 'void *' : unknown size

Динамическое распределение памяти.

При компиляции программы на C++ оперативная память компьютера, выделенная программе, разбивается на 4 области:

- область кода;
- область глобальных данных;
- область стека (stack);
- динамическая область (heap – куча).

Нижний адрес оперативной памяти – наименьший.
В области кода содержится программный код.
В области глобальных данных – глобальные переменные (объявленные вне любой функции) и переменные, объявленные как static.
Когда объявляются локальные переменные, они создаются в стеке, и при этом указатель стека двигается вниз. Когда область действия переменных заканчивается (выход из функции - локальная переменная больше не нужна и ее можно уничтожить, чтобы не занимала место), память автоматически освобождается путем смещения указателя стека вверх. Размер стековой памяти должен быть известен при компиляции. Стек занимает верхнюю часть области программы и растет вниз.
В программе возможно существование переменных, размер которых при компиляции неизвестен. Для них необходимо самостоятельно выделять память из свободной области. Свободная область занимает верхнюю часть памяти программы и растет вверх.
На переменные, находящиеся в heap, не действуют правила о зоне видимости. Эти переменные видны всегда (но вопрос – видны ли указатели на них?). Поэтому не забывайте освобождать ненужную динамическую память.
В языке C использовались библиотечные функции malloc() и free(). В C++ эти функции включены в ядро языка. Для работы с динамической памятью используются унарные операторы new и delete.
Оператор new выделяет блок памяти и возвращает указатель на первую ячейку памяти этого блока. Если new не в состоянии найти необходимое пространство свободной памяти, он вернет указатель NULL.
указатель = new тип;
указатель = new тип(значение);
Пример.

```
int *pi;
pi = new int;
float *pf = new float(-3.5);
```

Освобождение памяти:

```
delete указатель;
delete pf;
```

Ни с указателем, ни с блоком памяти ничего не происходит. Просто память становится свободной, а указатель теряет силу. Его разыменование может иметь непредсказуемые последствия. Автоматически указатель не обнуляется, и более безопасно сделать это вручную:

```
delete pf;
pf = NULL;//так грамотнее поступить с ненужным указателем, для которого применен оператор delete
```

Пример, позволяющий оценить расположение переменных в стеке и куче:

```
#include <iostream.h>
void main()
{
    int i1, i2, i3;
    int *ps1, *ps2, *ps3;
    ps1 = &(i1);
    ps2 = &(i2);
    ps3 = &i3;
    cout << "ps1 = " << ps1 << endl;
    cout << "ps2 = " << ps2 << endl;
    cout << "ps3 = " << ps3 << endl;
    int *ph1, *ph2, *ph3;
    ph1 = new int;
    ph2 = new int;
    ph3 = new int;
    cout << "ph1 = " << ph1 << endl;
    cout << "ph2 = " << ph2 << endl;
    cout << "ph3 = " << ph3 << endl;
```

```
}
//на консоль напечатает примерно:
ps1 = 0x0012FF7C
ps2 = 0x0012FF78
ps3 = 0x0012FF74
ph1 = 0x00320B00
ph2 = 0x00320B38
ph3 = 0x00320B70
Press any key to continue
```

Динамические массивы.

Становится возможным создавать массивы, размер которых будет известен лишь в процессе выполнения программы.

```
указатель = new тип массива[размер];
delete[] указатель;
int i = 10;
int m[i]; // так нельзя, на динамические объекты (в том числе динамические массивы) могут указывать только указатели, а если это статический массив, то размер его должен быть константой
int *pm;
pm = new int[i]; // так можно
delete[] pm; // удаляем массив, когда он уже не нужен
//пример работы с динамическим массивом
int size;
cout << "Vvedite razmer massiva: ";
cin >> size;
int *pmassiv = new int[size];
for (int i=0; i < size; i++)
{
	cout << "Vvedite element: ";
	cin >> *pmassiv[i];
}
```

Работать с таким массивом, как и с любыми данными в динамической памяти, можно только через указатели.

Обратите внимание, что операция sizeof с указателями дает несколько иной результат:

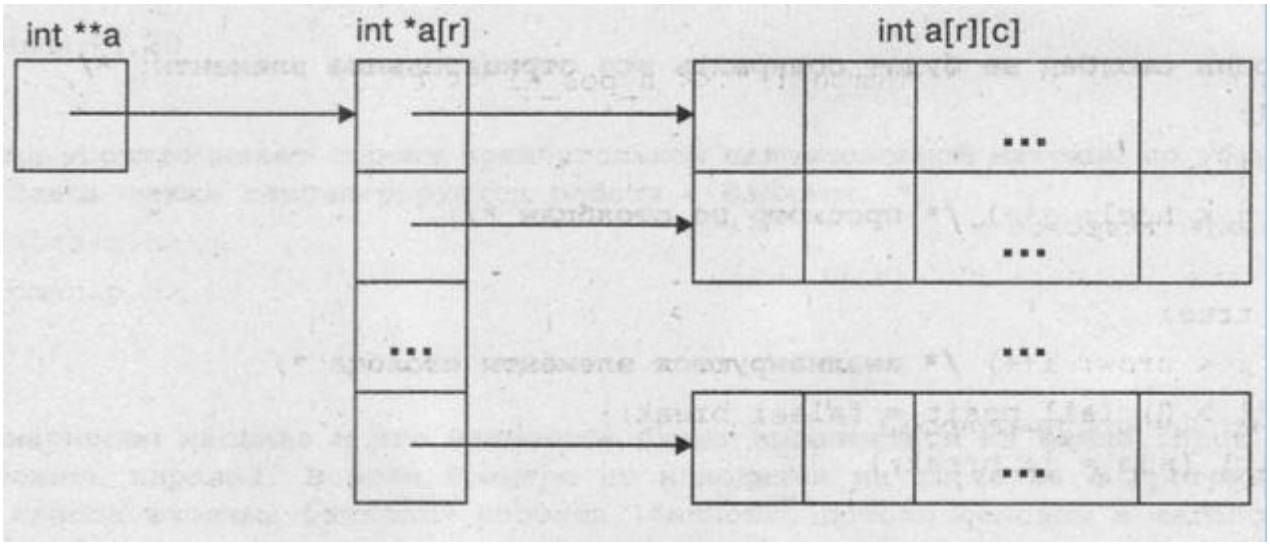
```
int m1[10];
cout << sizeof(m1) << endl; // выведет 40 (4*10) – размер всего массива
int *pi = m1;
cout << sizeof(*pi) << endl; // выведет 4 байта – размер одного целочисленного элемента массива
double *pm = new double[10];
cout << sizeof(pm) << endl; // выведет 4 (размер указателя)
cout << sizeof(*pm) << endl; // выведет 8 (размер double)
```

т.е. следить за размером массива должен программист, а использовать sizeof тут не стоит.

Динамические двумерные массивы.

Более универсальный и безопасный способ выделения памяти под двумерный массив, когда обе его размерности задаются на этапе выполнения программы, приведен ниже:

```
int nrow, ncol;
cout << " Введите количество строк и столбцов: ";
cin >> nrow >>ncol;
int**a =new int*[nrow]; // 1
for (int i = 0; i < nrow; i++) // 2
{
	a[i]=new int [ncol]; // 3
}
```



Выделение памяти под двумерный массив (указание количества строк и столбцов) происходит на этапе выполнения программы:

```
int r, c;
/* введите количество строк и столбцов массива */
cout << "Input the number of rows and columns: ";
cin >> r >> c;
/* здесь объявляется переменная типа "указатель на указатель на int" и выделяется память под массив указателей на строки массива */
int** a = new int*[r];
/* организуется цикл для выделения памяти под каждую строку массива */
for(int i = 0; i < r; i+ + )
/* Каждому элементу массива указателей на строки присваивается адрес начала участка памяти, выделенного под строку двумерного массива. Каждая строка состоит из ncol элементов типа int */
a[i] = new int[c];
```

Освобождение памяти из-под массива с любой размерностью выполняется так:

```
delete[] имя_массива;
```

Эта программа для прямоугольной целочисленной матрицы определяет номер самого левого столбца, содержащего только отрицательные элементы. Если же такого столбца нет, то выдает соответствующее сообщение.

```
#include <iostream>
#include <iomanip.h>//проверьте, требуется ли указать расширение для библиотечного файла, обеспечивающего манипуляции (форматирование) потока ввода-вывода (это ввод с клавиатуры и вывод на консоль)
void main()
{
	int nrow, ncol;
	/* введите количество строк и столбцов */
	cout << "Input the number of strings and columns: ";
	cin >> nrow >> ncol;
	int i, j;
	/* выделение памяти под массив */
	int** a = new int*[nrow];//в приведенных примерах обычно не проверяется успешность выделения ДООП, что нужно делать в программах: if(a == NULL){…}
	for( int i = 0; i < nrow; i++)
	{
		a[i] = new int[ncol];//тоже надо проверять успешность выделения ДООП
	}
	/* введите элементы массива */
	cout << "Input elements of the array: " << endl;
	for(int i =0; i < nrow; i++)
	{
		for(int j =0; j < ncol; j++)
		{
			cin >> a[i][ j];
		}
	}
	/* вывод массива */
```

```
	for(int i = 0; i < nrow; i++)
	{
		for (j =0; j < ncol; j++)
		{
			cout << setw(4) << a[i][j] << "\t";
		}
		cout << endl;
	}
	/* если ни один столбец не будет содержать все отрицательные элементы: */
	int num = -1;
	bool allPosit;
	for(int j =0; j < ncol; j++)/* просмотр по столбцам */
	{
		allPosit = true;
		for(int i =0; i < nrow; i++) /* анализируются элементы столбца */
		{
			if(a[i][j] > 0)
			{
				allPosit = false;
				break;
			}
			if(allPosit)//или allPosit == true
			{
				num = j;
				break;
			}
	}
	if(num == -1)
		cout << "There are no columns" << endl;/* таких столбцов нет */
	else
		cout << "The number of the column is: " << num << endl;
	delete[] a;
}//конец функции main()
```

Результаты выполнения программы:

```
Input the number of strings and columns: 3 4
Input elements of the array:
-5 2 0 1
6 2 -2 1
3 2 8 1
-5	2	0	1
6	2	-2	1
3	2	8	1
The number of the column is: 1
Press any key to continue
```

Не забывайте, что строки и столбцы нумеруются с нуля.

Указатели на структуры (для последующих лабораторных работ)

Указатель на структуру аналогичен указателю на любой другой тип данных.

```
struct coord_3D
{
    int x;
    int y;
    int z;
} point3, *pp3; // можно сразу же завести указатель
coord_3D *pp4; // или не сразу
pp3 = &point3;
coord_3D massiv[10]; //массив из десяти структур
pp4 = &massiv[0]; // указатель на массив структур
pp4 = massiv; // аналогично
```

Доступ к отдельным элементам структуры:

```
cout << (*pp4).x << endl; // взять структуру по адресу в pp4 и извлечь (прочитать) из нее элемент x
```

Внимание, следите за приоритетом!
cout << *pp4.x << endl; // взять элемент x структуры pp4 и разыменовать его (т.е. pp4 – имя структуры, а x – указатель, что ошибочно)
massiv[0].x=10;
(*pp1).x = 10; // аналогично, работает
pp1->x=10; // аналогично, работает
cout << pp4->y;
pp4++; // смещение на следующий элемент массива структур
Операции работы со структурами "->" и "." наряду со () для списка аргументов и [] для индексов имеют первый приоритет и, следовательно, связываются очень крепко. Если, например, имеется описание
struct Name
{//здесь, кстати, память под структуру не выделяется
	int x; // и так писать довольно глупо – мы завели указатель
	int *y; // на некую структуру, но самой структуры нет и быть не
} *p; // может – даже если завести еще одну такую – это другой тип
// допишем
int y0 = 100, y1 = 200, y2 = 300, y3 = 400;
struct Name2
{
	int x;
	int *y;
} s[4] = {10, &y0, 20, &y1, 30, &y2, 40, &y3},*p;
p = s;
cout << ++p->x << endl;	// 11 p - не изменился [0], извлекли x и ++
cout << (++p)->x << endl;	// 20	p - след.эл.массива [1], извлекли x
cout << (p++)->x << endl;	// 20	p - след.эл.массива [2] - но попозже, извлекли x
cout << *p->y++ << endl;	// 300 p - на [2], извлекли y, разыменовали y (y2) и нарастили y - на y1(но попозже)
cout << (*p->y)++ << endl;	// 200 p - на [2], извлекли y, разыменовали y(y1) и нарастили y1 = 201
cout << *p++->y << endl;	// 201 p нарастим, но попозже, извлекли y (y1), разыменовали

Указатели на константу.

Допустим, в программе имеется константа
const double PI = 3.1415;
Ее нельзя менять – на то она и константа.
Но что, если
double *pPI = &PI; // так нельзя
++ *pPI;
Отследить такое любому компилятору непросто.
Поэтому компилятор запрещает присваивать адреса констант обычным указателям.
const double *pPI = &PI; //указатель на константу
Обратите внимание – const стоит перед типом double (именно значение double нельзя менять)
*pPI = 3.14159; // ошибка
Но сам указатель – не константа, его можно менять.
const double E = 2.71828;
pPI = &E;
double g = 9.8; // можно и без const
pPI = &E;
Адрес константного объекта присваивается только указателю на константу. Вместе с тем, такому указателю может быть присвоен и адрес обычной переменной. Хотя g в примере выше и не является константой, компилятор не допустит изменения переменной g через pPI потому, что он не в состоянии определить, адрес какого объекта может содержать указатель в произвольный момент выполнения программы.

Константные указатели.

Существуют и константные указатели. Обратите внимание на разницу между константным указателем и указателем на константу! Константный указатель может адресовать как константу, так и переменную. Например:
int Nomer = 0;
int* const pNomer = &Nomer;
Здесь pNomer – константный указатель на неконстантный объект – целочисленную перменную. Это значит, что мы не можем присвоить ему адрес другого объекта, хотя сам объект допускает модификацию. Квалификатор const стоит перед именем pNomer – именно эту переменную нельзя изменять.
Вот как мог бы быть использован указатель pNomer:
pNomer = &PI; // ошибка – нельзя изменять сам указатель

*pNomer = 100; // можно изменять значение, которое находится по адресу, который содержится в указателе

Константный указатель на константу.

Константный указатель на константу является объединением двух вышерассмотренных случаев.

```
const double pi = 3.1452;
const double* const ptrPi = &pi;
```

Ни значение объекта, на который указывает ptrPi, ни значение самого указателя не может быть изменено в программе.

Примеры:

```
int i; // переменная i
int j = -1; // переменная j сразу проинициализирована
const int I1; // константа int – ошибка, ее нужно инициализировать сразу при создании (декларации)
const int J1 = j; // правильно
J1 = 5; //константу менять нельзя; нельзя даже пробовать присвоить ей значение после ее создания
int* const pI1; // константный указатель нужно сразу проинициализировать, ведь он тоже константа!
int* const pJ2 = &j; // константный указатель сразу инициализируется адресом обычной переменной (это правильно)
int* const pJ1 = &J1; // а инициализировать его адресом константы нельзя, поскольку это константный указатель на обычную целочисленную переменную, а не на константу
pJ2 = &i; // ошибка, сам константный указатель менять нельзя
*pJ2 = 5; // а значение по нему - менять можно
const int* pJ3; // указатель на константу – можно инициализировать позже
pJ3 = &J1; // сам указатель можно менять
pJ3 = &i; // ему не обязательно присваивать адрес константы
*pJ3 = 8; //значение через указатель на константу менять нельзя (даже если это на самом деле просто переменная i)
const int* const pI4; // константный указатель на целочисленную константу (его сразу нужно инициализировать, тут ошибка)
const int* const pI5= &i; // константный указатель на константу инициализируется адресом переменной
const int* const pJ4= &J1; // константный указатель на константу инициализируется адресом константы
pJ4 = &J1; // менять адрес в константном указателе нельзя
*pJ4 = 8; // менять через указатель на константу ее значение тоже нельзя
```

Функции и указатели.

В случае если функция должна менять свои аргументы, можно использовать указатели. Указатели также передаются по значению, внутри функции создается локальная переменная - указатель. Но так как этот указатель инициализируется адресом переменной из вызываемой программы, то эту переменную можно менять, используя этот адрес.

В качестве примера рассмотрим функцию, меняющую местами свои аргументы:

```
void swap (int* x, int* y)
{
	int t = *x;
	*x = *y;
	*y = t;
}
```

Обратиться к этой функции можно так:

```
int a = 3, b = 7;
swap (&a, &b);
```

Теперь a =7, и b=3.

Функция может иметь указатель не только в качестве аргумента, но и в качестве своего возвращаемого значения (тип функции).

Например, если функция принимает массив и возвращает другой массив (например, сумму строк):

```
double* sumstr ( int n, double* p[])
{
	double* ps = new double[n];
	for (int i=0; i< n; i++)
	{
		ps[i]=0;
		for (int j=0; j< n; j++)
		{
			ps[i] += p[i][j];
		}
	}
	return ps;
}
// вызов из main()
```

```
{//код
double* p1 = sumstr(4, ptr);
for (int i=0; i< 4; i++)
{
	cout << p1[i] << endl;
}
```
Обратите внимание, что в вызывающей функции объявлен только указатель, память под новый массив (одномерный) захватывается внутри функции.
Пример, когда функция возвращает двумерный массив – указатель на массив указателей:
```
#include <iostream>
int** singl(int n)
{
	int **p=new int*[n];
	/* Тип int*[n] - массив указателей на целые числа. Операция new возвращает указатель на выделенную память под этот массив и тип переменной p есть int**. Таким образом, p есть массив указателей на строки целых чисел будущей матрицы. */
	// цикл создания одномерных массивов - строк матрицы:
	for (int i=0; i< n; i++)
	{
		p[i]=new int[n];//в этих примерах не хватает проверки на успешность выделения ДООП
		for(int j=0; j< n; j++)
		{
			p[i][j] = (i == j ) ? 1 : 0;//используется тернарный оператор
		}
	}
	return p;
}
void main()
{
	int n;
	cout<< "Poriadok: " << endl;
	cin>>n;
	int** matr; //Указатель для формируемой матрицы
	matr = singl(n);
	for (int i=0; i< n; i++)
	{
		cout<< "\nStroka " << i << ": ";
		for (int j=0; j< n; j++)
		{
			cout << matr[i][j] << " ";
		}
	}
	for(int i=0; i< n; i++)
	{
		delete[] matr[i];
	}
	delete[] matr;
}
```
Передача массивов в функции.
Некоторую особенность имеет использование массивов в качестве аргументов. Эта особенность заключается в том, что имя массива преобразуется к указателю на его первый элемент, т.е. при передаче массива происходит передача указателя. По этой причине вызываемая функция не может отличить, относится ли передаваемый ей указатель к началу массива или к одному единственному объекту.
```
int summa(int array[], int size)
{
	int res = 0;
	for(int i = 0; i < size; i++)
	{
		res += array[i];
	}
```

```
	return res;
}
```
В заголовке int array[ ] можно заменить на int* array (и только так), а выражение в теле функции array[i] можно таким и оставить или заменить на *(array+i) или даже на *array++ (всего 3 варианта), т.к. array не является именем массива, и следовательно, не является константным указателем. К функции summa() в main'е можно обратиться так:
```
int mas[100];
for (int i = 0; i < 100; i++)
{
	mas[i] = 2*i + 1;
}
int j = summa(mas, 100);
```
Передача многомерных массивов.

Если функции передаётся двумерный массив, то описание соответствующего аргумента функции должно содержать количество столбцов; количество строк - несущественно, поскольку фактически передаётся указатель.
```
double sum(double m[][4], int n)
{
	cout << sizeof(m[0]); // 32 – мы можем увидеть переданную размерность
	double sum = 0;
	for (int i=0; i < sizeof(m[0])/sizeof(double); i++) // используем размерность массива
	{
		for (int j=0; j< n; j++)
		{
			sum +=m[i][j];
		}
	}
	return sum;
}
```
Но в этом случае мы имеем дело с фиксированным количеством столбцов (количество строк может быть любым).

Пример использования функции sum() в функции main:
```
double A[4][4] = { 10, 12, 14, 17,
		15, 13, 11, 0,
		-3, 5.1, 6, 6 ,
		2, 8, 3, 1 };
cout << sum(A, 4) << endl;
double B[2][4]={ 10, 12, 14, 17, 15, 13, 11, 0 };
cout << sum(B,2) << endl;
```
Если мы хотим иметь дело с любым массивом:
```
void trans(int n, double* p[]) // транспонирование любого квадратного массива
{
	double x;
	for(int i=0; i< n-1; i++)
	{
		for (int j=i+1; j< n; j++)
		{
			x=p[i][j];
			p[i][j]=p[j][i];
			p[j][i]=x;
		}
	}
}
```
"Кусочек" main'а:
```
double* ptr[] = {A[0], A[1], A[2], A[3]};
trans(4, ptr);
for(int i=0; i< n; i++)
{
	cout << "\n строка"<< (i+1)<< ": ";
	for(int j=0; j< n; j++)
	{
		cout<< "\t"<< A[i][j];
	}
```

```
	cout<< "\n";
}
```
Обратите внимание, что изменяется именно тот массив, который мы передаем (то есть изменяется оригинал).
Если массив не квадратный, потребуется переписать функцию trans(), чтобы передать в нее 2 измерения.
Функция может иметь указатель не только в качестве аргумента, но и в качестве возвращаемого значения (тип функции).
Например, если функция принимает массив и возвращает другой массив (например, сумму строк):
```
double* sumstr( int n, double* p[] )
{
	double* ps = new double[n];
	for(int i=0; i< n; i++)
	{
		ps[i] = 0;
		for(int j=0; j< n; j++)
		{
			ps[i] += p[i][j];
		}
	}
	return ps;
}
// вызов из main'a:
//предыдущий код
double* p1 = sumstr(4, ptr);
for (int i=0; i< 4; i++)
{
	cout << p1[i] << endl;
}
```
Обратите внимание, что в вызывающей функции объявлен только указатель, память под новый массив (одномерный) захватывается внутри функции.
Пример, когда функция возвращает двумерный массив – указатель на массив указателей:
```
#include <iostream>
int** singl(int n)
{
	int** p = new int*[n];
	/* Тип int*[n] - массив указателей на целые числа. Операция new возвращает указатель на выделенную память под этот массив и тип переменной p есть int**. Таким образом, p есть массив указателей на строки целых чисел будущей матрицы. */
	// цикл создания одномерных массивов - строк матрицы:
	for (int i=0; i< n; i++)
	{
		p[i] = new int[n];
		for(int j = 0; j < n; j++)
		{
			p[i][j] = (i == j ) ? 1 : 0;
		}
	}
	return p;
}
void main()
{
	int n;
	cout<< "Poriadok: " << endl;
	cin>>n;
	int** matr; //Указатель для формируемой матрицы
	matr = singl(n);
	for (int i=0; i< n; i++)
	{
		cout<< "\n stroka " << i << ": ";
		for(int j = 0; j < n; j++)
		{
```

```
                cout << matr[i][j] << '\t';
            }
        }
        for(int i=0; I < n; i++)
        {
            delete[] matr[i];
        }
        delete[] matr;
}
```

Задание 1

Выполняя данное задание нужно создать **одну** переменную, **один** указатель на нее и **одну** ссылку на эту переменную.

1	Создать целочисленную переменную. Создать указатель и ссылку на данную целочисленную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, большее на единицу ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
2	Создать переменную типа double. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, большее на единицу ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
3	Создать переменную типа long int. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, большее на единицу ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
4	Создать переменную типа float. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, в 2 раза большее ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
5	Создать переменную типа short int. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, меньшее на единицу ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
6	Создать переменную логического типа bool. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение «истина». Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, обратное (то есть противоположное) ее текущему значению, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
7	Создать переменную типа char. Проинициализировать переменную значением символ 'S'. Создать указатель и ссылку на данную переменную. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, большее на единицу ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку (во всех трех случаях прибавить 1). Распечатать новое значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку. Попробуйте распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке.

8	Создать беззнаковую целочисленную переменную. Создать указатель и ссылку на данную целочисленную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение, вводимое пользователем с клавиатуры. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, большее на двойку ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
9	Создать переменную типа long double. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение, вводимое пользователем с клавиатуры. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, большее на двойку ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
10	Создать переменную типа unsigned long int. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение, вводимое пользователем с клавиатуры. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, большее на двойку ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
11	Создать переменную типа long float. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение, вводимое пользователем с клавиатуры. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, в полтора раза большее ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
12	Создать переменную типа unsigned short int. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение, вводимое пользователем с клавиатуры. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, меньшее на двойку ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
13	Создать переменную логического типа bool. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение «ложь». Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, обратное (то есть противоположное) ее текущему значению, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
14	Создать переменную типа unsigned char. Проинициализировать переменную значением символ 'D'. Создать указатель и ссылку на данную переменную. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, большее на единицу ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку (во всех трех случаях прибавить 1). Распечатать новое значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку. Попробуйте распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке.
15	Создать знаковую целочисленную переменную. Создать указатель и ссылку на данную целочисленную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, большее на тройку ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
16	Создать переменную длинного вещественного типа. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение. Распечатать на консоль значение переменной,

	используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, большее на тройку ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
17	Создать переменную типа unsigned long int. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, большее на тройку ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
18	Создать переменную вещественного типа одинарной точности. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, в 2 с половиной раза большее ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
19	Создать переменную типа unsigned short int. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, меньшее на тройку ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
20	Создать переменную логического типа. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение, определяемое пользователем с клавиатуры (0 – это «ложь»; 1 – это «истина»). Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, обратное (то есть противоположное) ее текущему значению, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
21	Создать переменную типа signed char. Проинициализировать переменную значением символ 'F'. Создать указатель и ссылку на данную переменную. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, большее на единицу ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку (во всех трех случаях прибавить 1). Распечатать новое значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку. Попробуйте распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке.
22	Создать короткую целочисленную переменную. Создать указатель и ссылку на данную целочисленную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение, вводимое пользователем с клавиатуры. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, большее на семерку ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
23	Создать переменную вещественного типа двойной точности. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение, вводимое пользователем с клавиатуры. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, меньшее на единицу ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
24	Создать переменную типа signed long int. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение, вводимое пользователем с клавиатуры. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, большее на семерку ее текущего значения, используя имя

	переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
25	Создать переменную для хранения вещественного числа. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение, вводимое пользователем с клавиатуры. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, в 2 раза меньшее ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
26	Создать переменную типа signed short int. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение, вводимое пользователем с клавиатуры. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, меньшее на пятерку ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
27	Создать переменную типа bool. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение, определяемое пользователем с клавиатуры (0 – это «ложь»; 1 – это «истина»). Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, обратное (то есть противоположное) ее текущему значению, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
28	Создать переменную символьного типа. Проинициализировать переменную значением символ 'G'. Создать указатель и ссылку на данную переменную. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, большее на единицу ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку (во всех трех случаях прибавить 1). Распечатать новое значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку. Попробуйте распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке.
29	Создать беззнаковую длинную целочисленную переменную. Создать указатель и ссылку на данную целочисленную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, большее на сотню ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.
30	Создать знаковую короткую целочисленную переменную. Создать указатель и ссылку на данную переменную. Распечатать на консоль адрес переменной, значение адреса, который хранится в указателе и значение адреса, который хранится в ссылке. Присвоить переменной значение, вводимое пользователем с клавиатуры. Распечатать на консоль значение переменной, используя сначала имя переменной, потом – используя указатель, потом – используя ссылку. Присвоить переменной значение, меньшее на двойку ее текущего значения, используя имя переменной, потом – используя указатель, потом – используя ссылку. Распечатать значение переменной, используя имя переменной, потом – используя указатель, потом – используя ссылку.

Задание 2

(состоит из 5 подзаданий для каждого учащегося)

Вариант 1

Во всех заданиях использовать указатели и косвенную адресацию.

Задание 1.

В одномерном массиве из 100 элементов определить количество элементов между минимумом и максимумом.

m = { 16, 78, 99, 6, -29, 19, -52, 65, -88, 51,
-79, -22, 32, -25, -62, -69, -2, -59, -75, 89,
-87, 95, -22, 85, -49, -75, 76, 73, -59, -52,
30, 49, -28, -48, 0, 57, -6, -85, 0, -18,
-97, -21, -95, 64, 22, -2, 69, -84, -1, -71,
-25, 47, 72, 43, 15, -44, 44, 61, 4, 74,
88, -61, 0, -64, -83, 97, 0, 90, 15, 8,
-54, 19, 73, 35, -67, -87, 85, -99, -70, 10,

98, 58, -10, -29, 95, 62, 77, 89, 36, -32,
78, 60, -79, -18, 30, -13, -34, -92, 1, -38 }

Вывести полученное число.

Задание 2.

Из предыдущего одномерного сформировать двумерный массив [10][10]. Вывести адреса всех элементов, равных 0.

Задание 3.

Сформировать массив типа char размером в 10x10x10 элементов. Проинициализировать его случайными символами от A до Z. Вывести каждый седьмой элемент этого массива. Определить, сколько раз в полученной строке встретилась буква 'Q'.

Задание 4.

Пользователь вводит с клавиатуры строку – предложение с пробелами и знаками пунктуации длиной до 100 символов. Вывести на экран длину каждого слова.

Задание 5. Подготовьте ответы на контрольные вопросы.

1. Есть ли в коде ниже ошибки и как их исправить? Что будет выведено на экран?

```
int *a;
int b[2];
a = b;
b[0] = 7;
b[1] = 10;
*a++;
cout << *a;
```

2. Объявлен массив строк. Как присвоить указателю адрес третьей строки?

```
char lines[10][20];
char *pl;
```

a. pl = *lines[2];
b. pl = lines[2];
c. pl = &lines[2];
d. pl = *lines[2][0];
e. pl = lines[2][0];
f. pl = &lines[2][0];

3. Найдите (если есть) ошибки компиляции; укажите, как их исправить:

a.
```
int x[][2]= {1,2,3,4,5,6,7,8,9,10,11,12};
int *px = x;
```
b.
```
int a, *pb, *pc;
```
c.
```
int x = 5;
int *px = &(x-1);
```
d.
```
int x = 5, y = 3, z;
int *px = &x, *py = &y;
z = &px + &py;
```
e.
```
int x[10];
int *px = &x[0];
int *py = px++;
```
f.
```
int *px = &x[4], *py = &x[8];
cout << px>=py?"раньше":"позже";
```

Вариант 2

Во всех заданиях использовать указатели и косвенную адресацию.

Задание 1.

В одномерном массиве из 100 элементов определить сумму отрицательных элементов.

```
m = { 16, 78, 99, 6, -29, 19, -52, 65, -88, 51,
    -79, -22, 32, -25, -62, -69, -2, -59, -75, 89,
    -87, 95, -22, 85, -49, -75, 76, 73, -59, -52,
     30, 49, -28, -48, 0, 57, -6, -85, 0, -18,
    -97, -21, -95, 64, 22, -2, 69, -84, -1, -71,
    -25, 47, 72, 43, 15, -44, 44, 61, 4, 74,
     88, -61, 0, -64, -83, 97, 0, 90, 15, 8,
    -54, 19, 73, 35, -67, -87, 85, -99, -70, 10,
     98, 58, -10, -29, 95, 62, 77, 89, 36, -32,
```

78, 60, -79, -18, 30, -13, -34, -92, 1, -38 }

Вывести полученное число.

Задание 2.

Из предыдущего одномерного сформировать двумерный массив [20][5]. Вывести адреса всех строк.

Задание 3.

Сформировать массив типа char размером в 7x5x8 элементов. Проинициализировать его случайными символами от A до Z. Вывести строки, в которых встретилась буква 'W'. (Строки - это второй индекс справа, их всего 7*5).

Задание 4.

Пользователь вводит с клавиатуры строку – предложение с пробелами и знаками пунктуации длиной до 100 символов. Вывести на экран адреса всех пробелов.

Задание 5. Подготовьте ответы на контрольные вопросы.

1. Есть ли в коде ниже ошибки и как их исправить? Что будет выведено на экран?

```
int *a;
int b[2];
a = b;
b[0] = 7;
b[1] = 10;
*(++a);
cout << *a;
```

2. Объявлен массив строк. Как вывести на экран третью строку?

```
char lines[10][20];
char *pl = &lines[0][0];
```

a. cout << pl[3];
b. cout << *pl[3];
c. cout << pl+3;
d. cout << *pl[3][0];
e. cout << *pl+3;
f. cout << *(pl+3);

3. Найдите (если есть) ошибки компиляции; укажите, как их исправить:

a.
```
int x[][2]= {1,2,3,4,5,6,7,8,9,10,11,12};
int *px = &x;
```

b.
```
int a, &pb, &pc;
```

c.
```
int x = 5;
int *px = &(x);
```

d.
```
int x = 5, y = 3, z;
int *px = &x, *py = &y;
z = *px + *py;
```

e.
```
int x[10];
int *px = x;
int *py = px;
```

f.
```
int *px = &x[4], *py = &x[8];
cout << px==py?"равны":"не равны";
```

Вариант 3

Во всех заданиях использовать указатели и косвенную адресацию.

Задание 1.

В одномерном массиве из 100 элементов определить среднее арифметическое.

```
m = { 16,  78,  99,   6, -29,  19, -52,  65, -88,  51,
     -79, -22,  32, -25, -62, -69,  -2, -59, -75,  89,
     -87,  95, -22,  85, -49, -75,  76,  73, -59, -52,
      30,  49, -28, -48,   0,  57,  -6, -85,   0, -18,
     -97, -21, -95,  64,  22,  -2,  69, -84,  -1, -71,
     -25,  47,  72,  43,  15, -44,  44,  61,   4,  74,
      88, -61,   0, -64, -83,  97,   0,  90,  15,   8,
     -54,  19,  73,  35, -67, -87,  85, -99, -70,  10,
      98,  58, -10, -29,  95,  62,  77,  89,  36, -32,
      78,  60, -79, -18,  30, -13, -34, -92,   1, -38 }
```

Вывести полученное число.

Задание 2.

Из предыдущего одномерного сформировать двумерный массив [50][2]. Вывести среднее арифметическое для каждой строки.

Задание 3.

Сформировать массив типа char размером в 10x10x10 элементов. Проинициализировать его случайными символами от A до Z. Сформировать массив адресов тех элементов, где встретилась буква 'Z'. Вывести оба массива.

Задание 4.

Пользователь вводит с клавиатуры строку – предложение с пробелами и знаками пунктуации длиной до 100 символов. Вывести на экран адрес начала каждого слова.

Задание 5. Подготовьте ответы на контрольные вопросы.

1. Есть ли в коде ниже ошибки и как их исправить? Что будет выведено на экран?

```
int *a;
int b[2];
a = b;
b[0] = 7;
b[1] = 10;
cout << a;
```

2. Объявлен массив строк. Как присвоить пятому символу третьей строки значение 'R'?

```
char lines[10][20];
char *pl;
```

a. pl[5][3] = 'R';
b. *pl[5][3] = 'R';
c. &pl[5][3] = 'R';
d. pl[5]+3 = 'R';
e. *(pl+5*20+3) = 'R';
f. pl[5][3] = &('R');

3. Найдите (если есть) ошибки компиляции; укажите, как их исправить:

a.
```
int x[][2]= {1,2,3,4,5,6,7,8,9,10,11,12};
int *px = &x;
```
b.
```
int *pb;
*pc;
```
c.
```
int x = 5;
int *px = *x;
```
d.
```
int x = 5, y = 3, z;
int *px = &x, *py = &y;
z = px - py;
```
e.
```
int x[10];
int *px = &x[0];
int *py = *px++;
```
f.
```
int *px = &x[4], *py = &x[8];
cout << *px>=*py?"раньше":"позже";
```

Вариант 4

Во всех заданиях использовать указатели и косвенную адресацию.

Задание 1.

В одномерном массиве из 100 элементов определить количество чисел, кратных 10.

```
m = { 16,  78,  99,   6, -29,  19, -52,  65, -88,  51,
     -79, -22,  32, -25, -62, -69,  -2, -59, -75,  89,
     -87,  95, -22,  85, -49, -75,  76,  73, -59, -52,
      30,  49, -28, -48,   0,  57,  -6, -85,   0, -18,
     -97, -21, -95,  64,  22,  -2,  69, -84,  -1, -71,
     -25,  47,  72,  43,  15, -44,  44,  61,   4,  74,
      88, -61,   0, -64, -83,  97,   0,  90,  15,   8,
     -54,  19,  73,  35, -67, -87,  85, -99, -70,  10,
      98,  58, -10, -29,  95,  62,  77,  89,  36, -32,
      78,  60, -79, -18,  30, -13, -34, -92,   1, -38 }
```

Вывести полученное число.

Задание 2.

Из предыдущего одномерного сформировать двумерный массив [25][4]. Вывести максимум для каждой строки.

Задание 3.

Сформировать массив типа char размером в 3x7x12 элементов. Проинициализировать его случайными символами от A до Z. Вывести все гласные. Определить, сколько раз в элементах с последним индексом 10 встретилась буква 'Q'.

Задание 4.

Пользователь вводит с клавиатуры строку – предложение с пробелами и знаками пунктуации длиной до 100 символов. Вывести на экран индексы всех знаков пунктуации.

Задание 5. Подготовьте ответы на контрольные вопросы.

1. Есть ли в коде ниже ошибки и как их исправить? Что будет выведено на экран?

```
int *a;
int b[2];
a = b;
b[0] = 7;
b[1] = 10;
cout << *a+1;
```

2. Объявлен массив строк. Как в третью строку записать слово "monday"?

```
char lines[10][20];
char *pl= &lines[0][0];
```

a.
```
strcpy(pl+3, "monday");
cout << lines[3] << endl;
```

b.
```
strcpy(*pl+3, "monday");
cout << lines[3] << endl;
```

c.
```
strcpy(*(pl+3), "monday");
cout << lines[3] << endl;
```

d.
```
strcpy(pl+20*3, "monday");
cout << lines[3] << endl;
```

e.
```
strcpy(*pl+20*3, "monday");
cout << lines[3] << endl;
```

f.
```
strcpy(*(pl+20*3), "monday");
cout << lines[3] << endl;
```

3. Найдите (если есть) ошибки компиляции; укажите, как их исправить:

a.
```
int x[3][2];
int *px = x[0];
```

b.
```
float *pb, *pc = pb;
```

c.
```
int x = 5;
int *px = x;
```

d.
```
int x = 5, y = 3, z;
int *px = &x, *py = &y;
z = px + py;
```

e.
```
int x[10];
int *px = x;
int *py = px*2;
```

f.
```
int *px = &x[4], *py = &x[8];
cout << px!=py?"да":"нет";
```

Вариант 5

Во всех заданиях использовать указатели и косвенную адресацию.

Задание 1.

В одномерном массиве из 100 элементов определить сумму положительных элементов.

```
m = { 16,  78,  99,   6, -29,  19, -52,  65, -88,  51,
     -79, -22,  32, -25, -62, -69,  -2, -59, -75,  89,
     -87,  95, -22,  85, -49, -75,  76,  73, -59, -52,
      30,  49, -28, -48,   0,  57,  -6, -85,   0, -18,
     -97, -21, -95,  64,  22,  -2,  69, -84,  -1, -71,
     -25,  47,  72,  43,  15, -44,  44,  61,   4,  74,
      88, -61,   0, -64, -83,  97,   0,  90,  15,   8,
     -54,  19,  73,  35, -67, -87,  85, -99, -70,  10,
      98,  58, -10, -29,  95,  62,  77,  89,  36, -32,
      78,  60, -79, -18,  30, -13, -34, -92,   1, -38 }
```

Вывести полученное число.

Задание 2.

Из предыдущего одномерного сформировать двумерный массив [5][20]. Вывести адреса первого и последнего элементов каждой строки.

Задание 3.

Сформировать массив типа char размером в 4x11x3 элементов. Проинициализировать его случайными символами от A до Z. Вывести первые символы строк (строк всего 44).

Задание 4.

Пользователь вводит с клавиатуры строку – предложение с пробелами и знаками пунктуации длиной до 100 символов. Вывести на экран все знаки пунктуации.

Задание 5. Подготовьте ответы на контрольные вопросы.

1. Есть ли в коде ниже ошибки и как их исправить? Что будет выведено на экран?

```
int *a;
int b[2];
a = &b[1];
b[0] = 7;
b[1] = 10;
a--;
cout << *a;
```

2. Объявлен массив строк. Как вывести на экран третью строку?

```
char lines[10][20];
char *pl = lines[0];
```

a. `cout << pl+3;`
b. `cout << *(pl+3);`
c. `cout << pl + 3*10;`
d. `cout << pl + 3*20;`
e. `cout << *(pl+3*10);`
f. `cout << *(pl + 3*20);`

3. Найдите (если есть) ошибки компиляции; укажите, как их исправить:

a.
```
int x[][2]= {1,2,3,4,5,6,7,8,9,10,11,12};
int *px = x[1];
```
b.
```
int *pb = *pc;
```
c.
```
int x = 5;
int *px = &x-1;
```
d.
```
int x = 5, y = 3, z;
int px = x, py = y;
z = px + py;
```
e.
```
int x[10];
int *px = x;
int *py = px++;
```
f.
```
int *px = &x[4], *py = &x[8];
cout << px-py?"раньше":"позже";
```

Вариант 6

Во всех заданиях использовать указатели и косвенную адресацию.

Задание 1.

В одномерном массиве из 100 элементов определить количество отрицательных элементов.

```
m = { 16,  78,  99,   6, -29,  19, -52,  65, -88,  51,
     -79, -22,  32, -25, -62, -69,  -2, -59, -75,  89,
     -87,  95, -22,  85, -49, -75,  76,  73, -59, -52,
      30,  49, -28, -48,   0,  57,  -6, -85,   0, -18,
     -97, -21, -95,  64,  22,  -2,  69, -84,  -1, -71,
     -25,  47,  72,  43,  15, -44,  44,  61,   4,  74,
      88, -61,   0, -64, -83,  97,   0,  90,  15,   8,
     -54,  19,  73,  35, -67, -87,  85, -99, -70,  10,
      98,  58, -10, -29,  95,  62,  77,  89,  36, -32,
      78,  60, -79, -18,  30, -13, -34, -92,   1, -38 }
```

Вывести полученное число.

Задание 2.

Из предыдущего одномерного сформировать двумерный массив [20][5]. Вывести индексы минимального и максимального элементов.

Задание 3.

Сформировать массив типа char размером в 15x2x2 элементов. Проинициализировать его случайными символами от A до Z. Вывести каждый третий элемент этого массива. Определить, сколько раз в полученной строке встретилась буква 'V'.

Задание 4.

Пользователь вводит с клавиатуры строку – предложение с пробелами и знаками пунктуации длиной до 100 символов. Вывести на экран длину строки.

Задание 5. Подготовьте ответы на контрольные вопросы.

1. Есть ли в коде ниже ошибки и как их исправить? Что будет выведено на экран?

```
int *a;
int b[2];
a = b;
b[0] = 7;
b[1] = 10;
a+1;
cout << *a;
```

2. Объявлен массив строк. Как присвоить указателю адрес третьей строки?

```
char lines[10][20];
char *pl = lines[1];
```

a. pl += 2;
b. pl += 3;
c. pl += 10*2;
d. pl += 10*3;
e. pl += 20*2;
f. pl += 20*3;

3. Найдите (если есть) ошибки компиляции; укажите, как их исправить:

a.
```
int x[][2]= {1,2,3,4,5,6,7,8,9,10,11,12};
int *px = &x;
```
b.
```
int b, *pb = &b, *pc = &pb;
```
c.
```
int x = 5;
int *px = &(x);
```
d.
```
int x = 5, y = 3, z;
int *px = &x, *py = &y;
z = *(px + py);
```
e.
```
int x[10];
int *px = x;
int *py = px+5;
```
f.
```
int *px = &x[4], *py = &x[8];
cout << px-py?"да":"нет";
```

Вариант 7

Во всех заданиях использовать указатели и косвенную адресацию.

Задание 1.

В одномерном массиве из 100 элементов определить количество элементов между максимальным отрицательным и минимальным положительным (ноль не относится ни к тем, ни к другим).

```
m = { 16,  78,  99,  26, -29,  19, -52,  65, -88,  51,
     -79, -22,  32, -25, -62, -69, -42, -59, -75,  89,
     -87,  95, -22,  85, -49, -75,  76,  73, -59, -52,
      30,  49, -28, -48,   0,  57,  46, -85,   0, -18,
     -97, -21, -95,  64,  22, -12,  69, -84, -11, -71,
     -25,  47,  72,  43,  15, -44,  44,  61,  54,  74,
      88, -61,   0, -64, -83,  97,   0,  90,  15,  28,
     -54,  19,  73,  35, -67, -87,  85, -99, -70,  10,
      98,  58, -10, -29,  95,  62,  77,  89,  36, -32,
      78,  60, -79, -18,  30, -13, -34, -92,  61, -38 }
```

Вывести полученное число.

Задание 2.

Из предыдущего одномерного сформировать двумерный массив [10][10]. Вывести адреса всех элементов, лежащих на главной диагонали.

Задание 3.

Сформировать массив типа char размером в 12x5x3 элементов. Проинициализировать его случайными символами от A до Z. Определить, сколько раз в массиве встретились два одинаковых символа подряд.

Задание 4.

Пользователь вводит с клавиатуры строку – предложение с пробелами и знаками пунктуации длиной до 100 символов. Вывести на экран все заглавные буквы.

Задание 5. Подготовьте ответы на контрольные вопросы.

1. Есть ли в коде ниже ошибки и как их исправить? Что будет выведено на экран?

```
int *a;
int b[2];
a = b;
b[0] = 7;
b[1] = 10;
(*(b+1))++;
cout << *(a+1);
```

2. Объявлен массив строк. Как присвоить пятому элементу второй строки символ 'D'?

```
char lines[10][20];
char *pl= lines[0];
```

a. *(lines[2] + 5) = 'D';
b. *(lines[5] + 2) = 'D';
c. *(lines + 2 + 5) = 'D';
d. *pl[2][5] = 'D';
e. *(pl + 2*10 + 5) = 'D';
f. *(pl + 2*20 + 5) = 'D';

3. Найдите (если есть) ошибки компиляции; укажите, как их исправить:

a.
```
int x[][2]= {1,2,3,4,5,6,7,8,9,10,11,12};
int *px = x;
```
b.
```
int a, *pb, *pc;
```
c.
```
int x = 5;
int *px = &(x-1);
```
d.
```
int x = 5, y = 3, z;
int px = x, py = y;
z = px + py;
```
e.
```
int x[10];
int *px = &x[0];
int *py = px++;
```
f.
```
int *px = &x[4], *py = &x[8];
cout << px+py?"yes":"no";
```

Вариант 8

Во всех заданиях использовать указатели и косвенную адресацию.

Задание 1.

В одномерном массиве из 100 элементов определить количество элементов между первым и последним нулевыми элементами.

```
m = { 16,  78,  99,   6, -29,  19, -52,  65, -88,  51,
     -79, -22,  32, -25, -62, -69,  -2, -59, -75,  89,
     -87,  95, -22,  85, -49, -75,  76,  73, -59, -52,
      30,  49, -28, -48,   0,  57,  -6, -85,   0, -18,
     -97, -21, -95,  64,  22,  -2,  69, -84,  -1, -71,
     -25,  47,  72,  43,  15, -44,  44,  61,   4,  74,
      88, -61,   0, -64, -83,  97,   0,  90,  15,   8,
     -54,  19,  73,  35, -67, -87,  85, -99, -70,  10,
      98,  58, -10, -29,  95,  62,  77,  89,  36, -32,
      78,  60, -79, -18,  30, -13, -34, -92,   1, -38 }
```

Вывести полученное число.

Задание 2.

Из предыдущего одномерного сформировать двумерный массив [10][10]. Найти сумму всех элементов, лежащих выше главной диагонали.

Задание 3.

Сформировать массив типа char размером в 5x10x7 элементов. Проинициализировать его случайными символами от A до Z. Определить количество столбцов, не содержащих ни одной буквы 'Q' (всего столбцов 5*7).

Задание 4.

Пользователь вводит с клавиатуры строку – предложение с пробелами и знаками пунктуации длиной до 100 символов. Вывести на экран каждое третье слово.

Задание 5. Подготовьте ответы на контрольные вопросы.

1. Есть ли в коде ниже ошибки и как их исправить? Что будет выведено на экран?

```
int *a;
int b[2];
a = b;
b[0] = 7;
b[1] = 10;
(*a)++;
cout << *a;
```

2. Объявлен массив строк. Как присвоить указателю адрес третьей строки?

```
char lines[10][20];
char *pl;
```

a. pl = *lines[2];
b. pl = lines[2];
c. pl = &lines[2];
d. pl = *lines[2][0];
e. pl = lines[2][0];
f. pl = &lines[2][0];

3. Найдите (если есть) ошибки компиляции; укажите, как их исправить:

a.
```
int x[][2]= {1,2,3,4,5,6,7,8,9,10,11,12};
int *px = x;
```
b.
```
int a, *pb, *pc;
```
c.
```
int x = 5;
int *px = *(x-1);
```
d.
```
int x = 5, y = 3, z;
int *px = &x, *py = &y;
z = *px + *py;
```
e.
```
int x[10];
int *px = &x[0];
int *py = ++px;
```
f.
```
int *px = &x[4], *py = &x[8];
cout << px>=py?"yes":"no";
```

Вариант 9

Во всех заданиях использовать указатели и косвенную адресацию.

Задание 1.

В одномерном массиве из 100 элементов найти самую длинную цепочку отрицательных элементов.

```
m = { 16,  78,  99,   6, -29,  19, -52,  65, -88,  51,
     -79, -22,  32, -25, -62, -69,  -2, -59, -75,  89,
     -87,  95, -22,  85, -49, -75,  76,  73, -59, -52,
      30,  49, -28, -48,   0,  57,  -6, -85,   0, -18,
     -97, -21, -95,  64,  22,  -2,  69, -84,  -1, -71,
     -25,  47,  72,  43,  15, -44,  44,  61,   4,  74,
      88, -61,   0, -64, -83,  97,   0,  90,  15,   8,
     -54,  19,  73,  35, -67, -87,  85, -99, -70,  10,
      98,  58, -10, -29,  95,  62,  77,  89,  36, -32,
      78,  60, -79, -18,  30, -13, -34, -92,   1, -38 }
```

Вывести полученную цепочку.

Задание 2.

Из предыдущего одномерного сформировать двумерный массив [25][4]. Вывести адреса всех строк и их начальные элементы.

Задание 3.

Сформировать массив типа char размером в 7x12x2 элементов. Проинициализировать его случайными символами от A до Z. Вывести все элементы в диапазоне от 'K' до 'S'.

Задание 4.

Пользователь вводит с клавиатуры строку – предложение с пробелами и знаками пунктуации длиной до 100 символов. Вывести на экран количество букв.

Задание 5. Подготовьте ответы на контрольные вопросы.

1. Есть ли в коде ниже ошибки и как их исправить? Что будет выведено на экран?

```
int *a;
int b[2];
a = b;
b[0] = 7;
b[1] = 10;
cout << *a++;
```

2. Объявлен массив строк. Как присвоить указателю адрес третьей строки?

```
char lines[10][20];
char *pl;
```

a. pl = *lines[2];
b. pl = lines[2];
c. pl = &lines[2];
d. pl = *lines[2][0];
e. pl = lines[2][0];
f. pl = &lines[2][0];

3. Найдите (если есть) ошибки компиляции; укажите, как их исправить:

a.
```
int x[3][2];
int *px = x[0];
```
b.
```
float *pb, *pc = pb;
```
c.
```
int x = 5;
int *px = x;
```
d.
```
int x = 5, y = 3, z;
int *px = &x, *py = &y;
z = px + py;
```
e.
```
int x[10];
int *px = x;
int *py = px*2;
```
f.
```
int *px = &x[4], *py = &x[8];
cout << px!=py?"да":"нет";
```

Вариант 10

Во всех заданиях использовать указатели и косвенную адресацию.

Задание 1.

В одномерном массиве из 100 элементов определить количество элементов, чье значение больше 50.

```
m = { 16,  78,  99,   6, -29,  19, -52,  65, -88,  51,
     -79, -22,  32, -25, -62, -69,  -2, -59, -75,  89,
     -87,  95, -22,  85, -49, -75,  76,  73, -59, -52,
      30,  49, -28, -48,   0,  57,  -6, -85,   0, -18,
     -97, -21, -95,  64,  22,  -2,  69, -84,  -1, -71,
     -25,  47,  72,  43,  15, -44,  44,  61,   4,  74,
      88, -61,   0, -64, -83,  97,   0,  90,  15,   8,
     -54,  19,  73,  35, -67, -87,  85, -99, -70,  10,
      98,  58, -10, -29,  95,  62,  77,  89,  36, -32,
      78,  60, -79, -18,  30, -13, -34, -92,   1, -38 }
```

Вывести полученное число.

Задание 2.

Из предыдущего одномерного сформировать двумерный массив [10][10]. Вывести адреса всех элементов, чье значение больше 50.

Задание 3.

Сформировать массив типа char размером в 2x5x12 элементов. Проинициализировать его случайными символами от A до Z. Выполнить операцию сравнения на равенство между элементами (буквами, кодами букв) на страницах этого массива (страниц всего 2 - это первый индекс). Результат (если пара сравниваемых букв одинаковы, то оставить их неизменными, иначе – заменить первую букву (букву, относящуюся к первой странице) на вторую букву (букву со второй страницы)) поместить в первой половине массива (то есть на первой странице) и вывести на экран в виде

двумерного массива 5x12.

Задание 4.

Пользователь вводит с клавиатуры строку – предложение с пробелами и знаками пунктуации длиной до 100 символов. Вывести на экран длину каждого слова.

Задание 5. Подготовьте ответы на контрольные вопросы.

1. Есть ли в коде ниже ошибки и как их исправить? Что будет выведено на экран?

```
int *a;
int b[2];
a = &b[1];
b[0] = 7;
b[1] = 10;
a--;
cout << *a;
```

2. Объявлен массив строк. Как вывести на экран третью строку?

```
char lines[10][20];
char *pl = lines[0];
```

a. `cout << pl+3;`
b. `cout << *(pl+3);`
c. `cout << pl + 3*10;`
d. `cout << pl + 3*20;`
e. `cout << *(pl+3*10);`
f. `cout << *(pl + 3*20);`

3. Найдите (если есть) ошибки компиляции; укажите, как их исправить:

a.
```
int x[][2]= {1,2,3,4,5,6,7,8,9,10,11,12};
int *px = x[1];
```
b.
```
int *pb = *pc;
```
c.
```
int x = 5;
int *px = &x-1;
```
d.
```
int x = 5, y = 3, z;
int px = x, py = y;
z = px + py;
```
e.
```
int x[10];
int *px = x;
int *py = px++;
```
f.
```
int *px = &x[4], *py = &x[8];
cout << px-py?"раньше":"позже";
```