

# \* Указатели на объекты Определения

- **Определение:** Указатель С это переменная, значением которой может быть либо адрес некоторого объекта - переменной, константы, массива, функции, либо пустой адрес - NULL или 0 - ноль.

Формы объявлений для указателей на переменные, константы или массивы типа ТИП:

## 1. не константные указатели на не константы:

- ТИП \* имя\_указателя;

## 2. не константные указатели на константы:

const ТИП \* имя\_указателя;

## 3. константные указатели на не константы:

ТИП \* const имя\_указателя;

## 4. константные указатели на константы:

const ТИП \* const имя\_указателя;

## \* На функции:

- \* Тип\_функции (\*имя\_Указателя)(список\_форм\_параметров);

# \* Указатели. Операции с указателями

Допустимы следующие основные действия с указателями:

1. объявления;
2. присваивания;
3. раскрытия ссылки - операция разыменованние указателя;
4. получения адреса указателя;
5. унарные операции инкремента и декремента изменения значения указателя;
6. аддитивные операции сложения и вычитания значения указателей и целых величин;
7. операции сравнения содержимого указателей.

**Примеры объявления и присваивания:**

```
char *c,ch='A';  
int *i,*k,ii,kk,date=2020;  
float *f,ff=123.456;  
i=&date; /*однотипное присваивание*/  
*i=date; /* раскрытие ссылки слева*/  
kk=*k; /*раскрытие ссылки справа*/  
i=k; /*однотипное присваивание */  
c=NULL; /*присваивание пустого адреса*/
```

# \* Указатели. Операции с указателями

## Примеры инициализации

```
int a=3;
```

```
int* b=&a;// Инициализация указателя адресом целой переменной
```

```
int *c(&a);// Инициализация указателя адресом целой переменной
```

```
int *d=c;// Инициализация указателя другим указателем
```

```
int* j = NULL;//Допустимо стандартом 1998
```

```
int*k=0; //Допустимо стандартом 1998
```

## Примеры контроля типа указателя

```
int c=10;
```

```
const int d=20;
```

```
int *const p1 = &c;//OK
```

```
int *const p2 = &d;//NOK
```

```
const int* p3=&d;//OK
```

```
int* p4=&d;//NOK
```

```
p4=&c;//OK
```

# \* Указатели. Операции с указателями

## Примеры приведения типа указателя

```
char *c;
```

```
int *k;
```

```
c=(char *)k; /*c указывает на целое как на символ*/
```

```
int a=6;
```

```
int *pa =&a;//Допустимо. Слева и справа от = адрес целого числа
```

```
void* vpa = (void*)&a;//Преобразование int* в void*
```

```
void* vp= (void*)pa;//Преобразование int* в void* .
```

## Пример определения адреса указателя

```
char *c;
```

```
void *a;
```

```
a=&c;
```

# \* Унарные и аддитивные операции с указателями

$ptr1 - ptr2$  вычисление числа элементов между  $ptr2$  и  $ptr1$

$ptr1 + int\_val$  вычисление указателя смещенного вверх

$ptr1 - int\_val$  вычисление указателя смещенного вниз

## Пример – разность указателей

```
#include <stdio.h>
```

```
#include <stddef.h>
```

```
void main()
```

```
{
```

```
int x[5]; /* массив из 5 элементов */
```

```
int *i, *k; /* */
```

```
ptrdiff_t j;
```

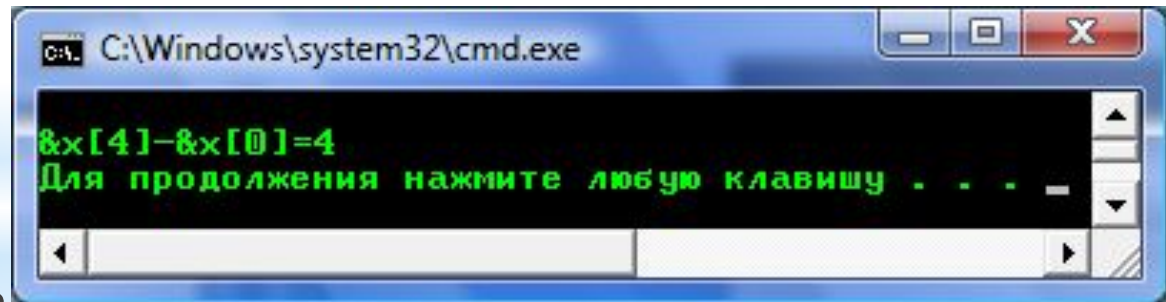
```
i=&x[0];
```

```
k=&x[4];
```

```
j=k-i;
```

```
printf("\n&x[4]-&x[0]=%d \n", (int)j); /* явное преобразование типа */
```

```
}
```



```
C:\Windows\system32\cmd.exe
&x[4]-&x[0]=4
Для продолжения нажмите любую клавишу . . . -
```

# \* Унарные и аддитивные операции с указателями

Пример – унарные операции с указателями

```
int main()
{
    int arr[4] = {10,20,30,40}, *parr=arr;
    int x;
    x=*parr;//1
    x=*parr++;//2
    parr = arr;
    x=++*parr;//3
    x=++*parr++;//4
    x=++(*parr);//5
    x=*++parr;    //6
    x=*parr++;//7
    x=++(*parr);//8
    x=(*parr)++;//9
    return 0;
}
```



# \* Унарные и аддитивные операции с указателями

```
C:\Windows\system32\cmd.exe

x = 10  parr = 0012FF34
arr[0]=10    arr[1]=20    arr[2]=30    arr[3]=40
x = 10  parr = 0012FF38
arr[0]=10    arr[1]=20    arr[2]=30    arr[3]=40
x = 11  parr = 0012FF34
arr[0]=11    arr[1]=20    arr[2]=30    arr[3]=40
x = 12  parr = 0012FF38
arr[0]=12    arr[1]=20    arr[2]=30    arr[3]=40
x = 21  parr = 0012FF38
arr[0]=12    arr[1]=21    arr[2]=30    arr[3]=40
x = 30  parr = 0012FF3C
arr[0]=12    arr[1]=21    arr[2]=30    arr[3]=40
x = 30  parr = 0012FF40
arr[0]=12    arr[1]=21    arr[2]=30    arr[3]=40
x = 41  parr = 0012FF40
arr[0]=12    arr[1]=21    arr[2]=30    arr[3]=41
x = 41  parr = 0012FF40
arr[0]=12    arr[1]=21    arr[2]=30    arr[3]=42

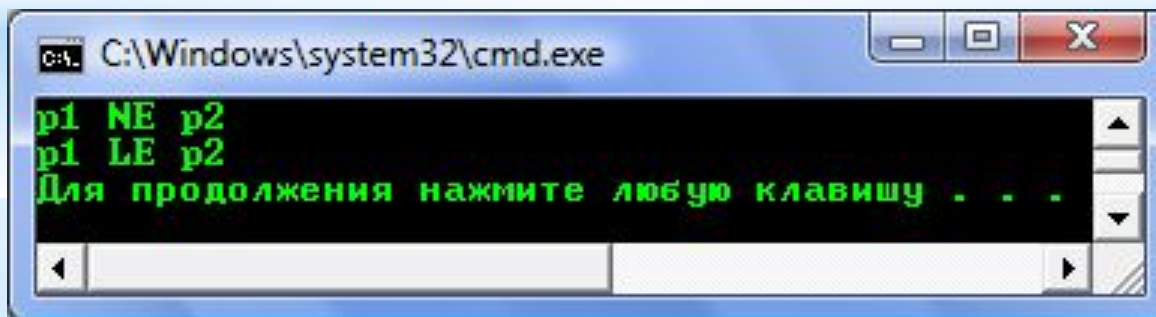
Для продолжения нажмите любую клавишу . . .
```

# \* Операции сравнения указателей

<code>ptr1 == ptr2</code>	сравнение на равенство
<code>ptr1 != ptr2</code>	сравнение на неравенство
<code>ptr1 &lt; ptr2</code>	сравнение на меньше
<code>ptr1 &lt;= ptr2</code>	сравнение на меньше и равно
<code>ptr1 &gt; ptr2</code>	сравнение на больше
<code>ptr1 &gt;= ptr2</code>	сравнение на больше и равно

## Пример:

```
#include <stdio.h>
int main(i)
{
    int x[ ]={1,2,3,4,5};
    int *p1,*p2;
    p1=x;
    p2=&x[2];
    if(p1!=p2) printf("p1 NE p2\n");
    if(p1>=p2) printf("p1 GE p2\n");
    if(p1<=p2) printf("p1 LE p2\n");
    return 0;
}
```



```
C:\Windows\system32\cmd.exe
p1 NE p2
p1 LE p2
Для продолжения нажмите любую клавишу . . .
```



## Доступ к элементам массивов

### 1. Одномерный массив

$E1[E2] \rightarrow *(E1 + E2)$ . Здесь  $E1$  – имя массива,  $E2$  – целое.

### 2. Многомерный массив

$E[i][j][k] \rightarrow *(E[i][j] + k) \rightarrow (*(E[i] + j) + k) \rightarrow (*(E + i) + j) + k$

# \* Указатели и динамическая память

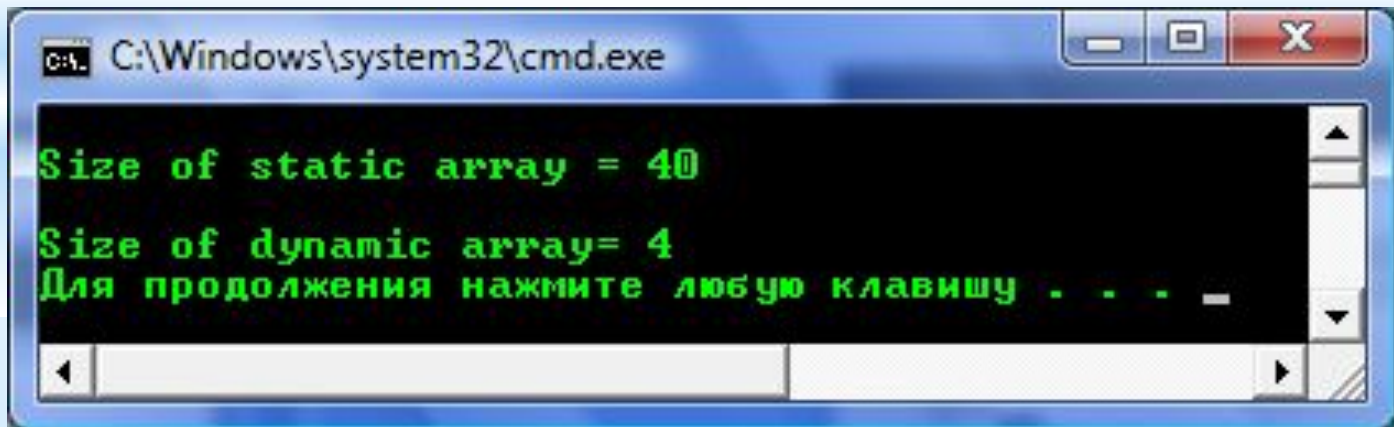
Функции С для работы с динамической памятью

<i>Функция</i>	<i>Прототип</i>
<b>malloc()</b>	<b>void * malloc(unsigned s);</b>
<b>calloc()</b>	<b>void * calloc(unsigned n, unsigned m);</b>
<b>realloc()</b>	<b>void * realloc(void *block, unsigned ns);</b>
<b>free()</b>	<b>void * free(void *block);</b>

# \* Статические и динамические массивы

## Пример:

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
int m_static[10],i,j,k;
int *m_dynamic = (int *) malloc (10*sizeof(int));
printf("\nSize of static array = %d\n",sizeof(m_static));
printf("\nSize of dynamic array= %d\n", sizeof(m_dynamic));
free(m_dynamic);
}
```



```
C:\Windows\system32\cmd.exe
Size of static array = 40
Size of dynamic array= 4
Для продолжения нажмите любую клавишу . . .
```

# \* Указатели и динамическая память

Пример:

```
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char* argv[])
{
    int x1[ ]={1,2,3,4,5};
    int x2[ ]={1,2,3,4,5,6};
    int x3[ ]={1,2};
    int *p1,*p2,*p3;
    p1=(int*)malloc(sizeof(x1));
    printf("\n p1= %p\n",p1);
    p2=(int*)realloc((void*)p1,sizeof(x2));
    printf("\n p2= %p\n",p2);
    p3=(int*)realloc((void*)p2,sizeof(x3));
    printf("\n p3= %p\n",p3);
    for(int i=0;i<sizeof(x3)/sizeof(x3[0]);i++)
    {
        p3[i]=x3[i];
        printf("\n p3[%d]=%d\n",i,p3[i]);
    }
    return 0;
}
```

```
p1= 00966F58
p2= 00966F58
p3= 00966F58
p3[0]=1
p3[1]=2
```

## \* Указатели и динамическая память

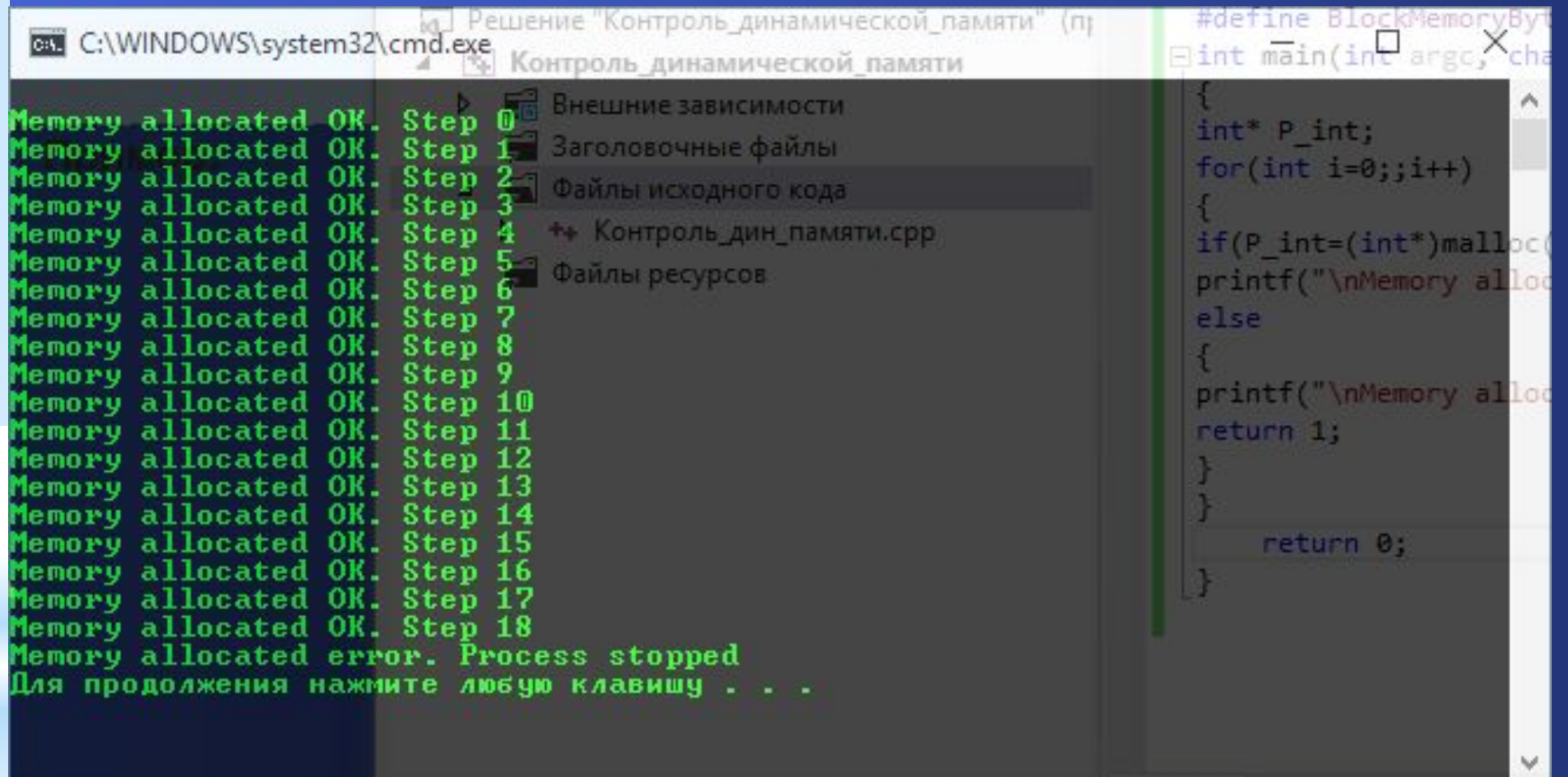
Пример:

```
#include <stdlib.h>
#include <stdio.h>
#define BlockMemoryByte 100000000
int main(int argc, char* argv[])
{
    int* P_int;
    for(int i=0;;i++)
    {
        if(P_int=(int*)malloc(BlockMemoryByte))
            printf("\nMemory allocated OK. Step %d",i);
        else
        {
            printf("\nMemory allocated error. Process stopped\n");
            return 1;
        }
    }
    return 0;
}
```



# \* Указатели и динамическая память

Пример:



The screenshot shows a Visual Studio IDE with a debugger window open. The console window on the left displays the following output:

```
Memory allocated OK. Step 0
Memory allocated OK. Step 1
Memory allocated OK. Step 2
Memory allocated OK. Step 3
Memory allocated OK. Step 4
Memory allocated OK. Step 5
Memory allocated OK. Step 6
Memory allocated OK. Step 7
Memory allocated OK. Step 8
Memory allocated OK. Step 9
Memory allocated OK. Step 10
Memory allocated OK. Step 11
Memory allocated OK. Step 12
Memory allocated OK. Step 13
Memory allocated OK. Step 14
Memory allocated OK. Step 15
Memory allocated OK. Step 16
Memory allocated OK. Step 17
Memory allocated OK. Step 18
Memory allocated error. Process stopped
Для продолжения нажмите любую клавишу . . .
```

The source code editor on the right shows the following C++ code:

```
#define BlockMemoryByt
int main(int argc, cha
{
    int* P_int;
    for(int i=0;;i++)
    {
        if(P_int=(int*)malloc(
        printf("\nMemory alloc
        else
        {
            printf("\nMemory alloc
            return 1;
        }
    }
    return 0;
}
```

# Многомерные динамические массивы

```
#define L1 4
#define L2 3
#define L3 2
#include <stdio.h>
#include <stdlib.h>
//
// A[L1*L2]*B[L2*L3] = C[L1*L3]
//
// C(i,j) = SCALAR_PROD(A[i]*B'[j])
//
int main()
{
double** A = (double**)malloc(L1 * sizeof(double*));
for (int i = 0; i < L1; i++) A[i] = (double*)malloc(L2 * sizeof(double));
double** B = (double**)malloc(L2 * sizeof(double*));
for (int i = 0; i < L2; i++) B[i] = (double*)malloc(L3 * sizeof(double));
double** C = (double**)malloc(L1 * sizeof(double*));
for (int i = 0; i < L1; i++) C[i] = (double*)malloc(L3 * sizeof(double));
```

# Многомерные динамические массивы

```
for (int i = 0; i < L1; i++)
{
    printf_s("Input %d elements of %d row A matrix\n", L2, i);
    for (int j = 0; j < L2; j++)
        scanf_s("%lf", &A[i][j]);
}
for (int i = 0; i < L2; i++)
{
    printf_s("Input %d elements of %d row B matrix\n", L3, i);
    for (int j = 0; j < L3; j++)
        scanf_s("%lf", &B[i][j]);
}
for (int i = 0; i < L1; i++)
    for (int j = 0; j < L3; j++)
    {
        double S = 0;
        for (int k = 0; k < L2; k++) S += A[i][k] * B[k][j];
        C[i][j] = S;
    }
```

# Многомерные динамические массивы

```
printf("C matrix\n");
for (int i = 0; i < L1; i++)
{
    printf_s("\n");
    for (int j = 0; j < L3; j++)
    {
        printf("C(%d, %d) = %lf ", i, j, C[i][j]);
    }
}
for (int i = 0; i < L1; i++) free(A[i]); free(A);
for (int i = 0; i < L2; i++) free(B[i]); free(B);
for (int i = 0; i < L1; i++) free(C[i]); free(C);
return 0;
return 0;
}
```

# Многомерные динамические массивы

```
Консоль отладки Microsoft Visual Studio
3.3
Input 3 elements of 1 row A matrix
1.1
2.2
3.3
Input 3 elements of 2 row A matrix
3.3
4.4
2.2
Input 3 elements of 3 row A matrix
5.5
6.6
7.7
Input 2 elements of 0 row B matrix
1.1
2.2
Input 2 elements of 1 row B matrix
3.3
4.4
Input 2 elements of 2 row B matrix
4.4
6.6
C matrix
C(0, 0) = 88.990000 C(0, 1) = 121.880000
C(1, 0) = 22.990000 C(1, 1) = 33.880000
C(2, 0) = 27.830000 C(2, 1) = 41.140000
C(3, 0) = 61.710000 C(3, 1) = 91.960000
C:\Users\Mihai\source\peros\Первое приложение C\Debug\Первое приложение C.exe (процесс 3696) завершает работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу...
```



# Многомерные динамические массивы

```
#include <stdio.h>
#include <stdlib.h>
#define L1 3
#define L2 4
#define L3 5
int main()
{
int i=0, j =0, k =0;
int *** A;
A=(int***)malloc(L1*sizeof(int**));

for(i=0; i<L1; i++)
A[i]=(int**)malloc(L2*sizeof(int*));
for(i=0; i<L1; i++)
for(j=0; j<L2; j++)
A[i][j]= (int*)malloc(L3*sizeof(int));
//Write to 3D matrix DATA
for(i=0; i<L1;i++)
for(j=0; j<L2;j++)
for(k=0; k<L3;k++)
A[i][j][k]= i*j*k;

for(k=0; k<L3;k++)
{
for(i=0; i<L1;i++)
{
for(j=0; j<L2;j++)
printf("A[%d][%d][%d] = %d\t",i,j,k,A[i][j][k]);
printf("\n");
}
}
return 0;
}
```

# \* Многомерные динамические массивы

```
malloc(L1*sizeof(int*));
int** A;
for(i=0; i<L1; i++)
    A[i] = (int**)malloc(L2*sizeof(int*));
for(i=0; i<L1; i++)
    for(j=0; j<L2; j++)
        A[i][j] = (int*)malloc(L3*sizeof(int));
for(i=0; i<L1; i++)
    for(j=0; j<L2; j++)
        for(k=0; k<L3; k++)
            A[i][j][k] = i*j*k;
for(i=0; i<L1; i++)
    for(j=0; j<L2; j++)
        for(k=0; k<L3; k++)
            printf("A[%d][%d][%d] = %d\n", i, j, k, A[i][j][k]);
printf("\n");
```

A[0][0][0] = 0 A[0][1][0] = 0 A[0][2][0] = 0 A[0][3][0] = 0  
A[1][0][0] = 0 A[1][1][0] = 0 A[1][2][0] = 0 A[1][3][0] = 0  
A[2][0][0] = 0 A[2][1][0] = 0 A[2][2][0] = 0 A[2][3][0] = 0  
A[0][0][1] = 0 A[0][1][1] = 0 A[0][2][1] = 0 A[0][3][1] = 0  
A[1][0][1] = 0 A[1][1][1] = 1 A[1][2][1] = 2 A[1][3][1] = 3  
A[2][0][1] = 0 A[2][1][1] = 2 A[2][2][1] = 4 A[2][3][1] = 6  
A[0][0][2] = 0 A[0][1][2] = 0 A[0][2][2] = 0 A[0][3][2] = 0  
A[1][0][2] = 0 A[1][1][2] = 2 A[1][2][2] = 4 A[1][3][2] = 6  
A[2][0][2] = 0 A[2][1][2] = 4 A[2][2][2] = 8 A[2][3][2] = 12  
A[0][0][3] = 0 A[0][1][3] = 0 A[0][2][3] = 0 A[0][3][3] = 0  
A[1][0][3] = 0 A[1][1][3] = 3 A[1][2][3] = 6 A[1][3][3] = 9  
A[2][0][3] = 0 A[2][1][3] = 6 A[2][2][3] = 12 A[2][3][3] = 18  
A[0][0][4] = 0 A[0][1][4] = 0 A[0][2][4] = 0 A[0][3][4] = 0  
A[1][0][4] = 0 A[1][1][4] = 4 A[1][2][4] = 8 A[1][3][4] = 12  
A[2][0][4] = 0 A[2][1][4] = 8 A[2][2][4] = 16 A[2][3][4] = 24

Для продолжения нажмите любую клавишу . . .