# 6. ОПЕРАЦИИ ДЛЯ РАБОТЫ С ДИНАМИЧЕСКОЙ ПАМЯТЬЮ

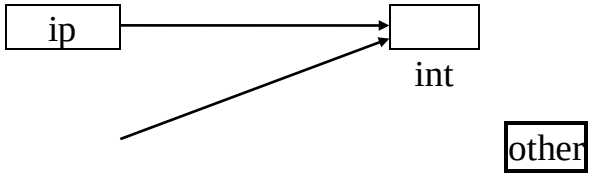
## 6.1. Операция выделения памяти *new*

Часто выражение, содержащее операцию new, имеет следующий вид: указатель_на_тип_= new имя_типа (инициализатор)

Инициализатор – это необязательное инициализирующее выражение, которое может использоваться для всех типов, кроме массивов. При выполнении оператора int *ip = new int;

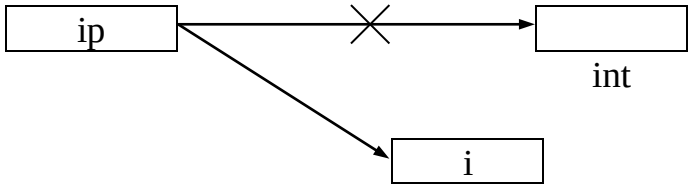
создаются 2 объекта: динамический безымянный объект и указатель на него с именем ip, значением которого является адрес динамического объекта. Можно создать и другой указатель на тот же динамический объект:

int *other = ip;




Если указателю ip присвоить другое значение, то можно потерять доступ к динамическому объекту: int *ip = new(int);

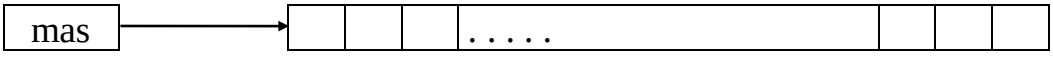
int i = 0; ip
= &i;




Теперь динамический объект по-прежнему будет существовать, но обратится к нему уже нельзя. Такие объекты называются мусором. При выделении памяти объект можно инициализировать: int *ip = new int(3);

Можно динамически распределить память и под массив:

double *mas = new double [50];




Теперь с этой динамически выделенной памятью можно работать как с обычным массивом:

```
*(mas + 5) = 3.27; mas[6] =
mas[5] + sin(mas[5]);
```

В случае успешного завершения операция new возвращает указатель со значением, отличным от нуля.

По умолчанию, если операция new не сработала (т. е. не найден непрерывный свободный фрагмент памяти нужного размера и память не выделилась), то генерируется так называемое ***исключение*** bad_alloc. Если это исключение будет неправильно обработано, то программа просто прекратит своё выполнение (произойдёт сбой) с ошибкой необработанного исключения.

Иногда процесс генерации исключения операцией new (как и сбой программы) нежелателен, поэтому есть альтернативная форма этой операции, которая возвращает нулевой указатель (**NULL** или **nullptr**), если память не может быть выделена. Нужно просто добавить константу std::nothrow между ключевым словом new и типом данных: int *ptr = new(std::nothrow) int;

Здесь указатель ptr станет нулевым, если динамическое выделение памяти не произойдет.

Поэтому хорошей практикой является проверка всех запросов на выделение памяти для обеспечения того, что эти запросы будут выполнены успешно и память выделится: int *ptr = new(std::nothrow) int;

```
if(!ptr){ cout << "Could not allocate
memory";  exit(1); // Например так!
}
```

## 6.2. Операция освобождения памяти *delete*

Операция **delete** освобождает для дальнейшего использования в программе участок памяти, ранее выделенной операцией **new**:

```
delete ip;                  // Удаляет динамический объект типа int,
// если было ip = new int; delete []mas;      //                  удаляет
динамический массив длиной 50,
// если было double *mas = new double[50];
```

Совершенно безопасно применять операцию к указателю NULL. Результат же повторного применения операции delete к одному и тому же указателю не определен. Обычно происходит ошибка, приводящая к зацикливанию.

Чтобы избежать подобных ошибок, можно применять следующую конструкцию:

```
int *ip = new int[500];
. . .
if(ip){delete []ip; ip = NULL;} else { cout <<"
память уже освобождена \n"; }
```

Пример:

Распределить память для матрицы из m строк и n столбцов: int

m, n;

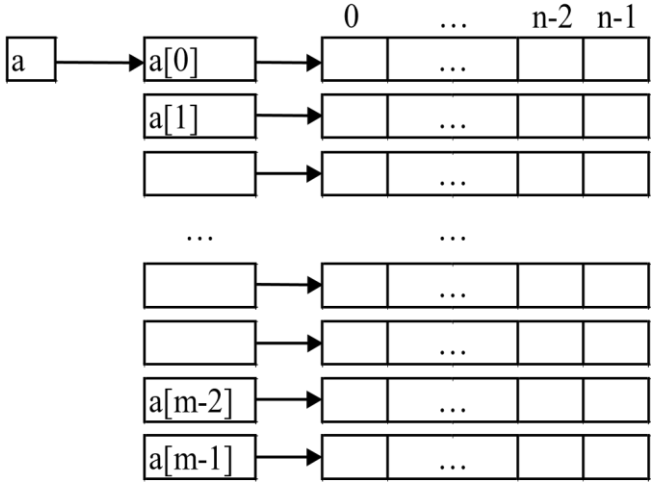
cout << "Задайте число строк и столбцов матрицы: \n"; cin >> m >> n; double **a = new double *[m]; // Массив из m указателей на double for (int i = 0; i < m; i++ ) // Распределяется строка матрицы:

```
if((a[i] = new(std::nothrow) double[n]) == NULL)
{ cout << "Нет памяти!\n";
exit(1);  }
```

Теперь к элементам этой матрицы можно обращаться обычным образом:

a[i][j] или *(a[i] + j) или *(*(a + i) + j)

Изобразить распределение памяти, соответствующее вышеприведенному фрагменту, можно следующим образом:




Освободить память здесь можно так:

```
for (i = 0; i < m; i++) delete []a[i];
delete []a;
```

Или так:

```
for(i = 0; i < m; i++){
```

```
delete []a[i];
a[i] = NULL;
} delete []a;
```