Оглавление

[1. Что такое указатель? 2](#_Toc62057995)

[2. Как определить указатель? 2](#_Toc62057996)

[3. Как проинициализировать указатель? 2](#_Toc62057997)

[4. Какие способы инициализации указателей вы знаете? 2](#_Toc62057998)

[5. Как обратиться к переменной, адресуемой указателем? 3](#_Toc62057999)

[6. Как получить адрес объекта? 3](#_Toc62058000)

[7. Какие операции можно выполнять над указателями? 3](#_Toc62058001)

[8. Что такое ссылка? 4](#_Toc62058002)

[9. Как объявить ссылку? 4](#_Toc62058003)

[10. Как создать переменную в динамической памяти? 4](#_Toc62058004)

[11. Как удалить переменную из динамической памяти? 4](#_Toc62058005)

[12. На что следует обращать особое внимание при работе с динамической памятью? 4](#_Toc62058006)

[13. Какие способы обращения к элементам массива вы знаете? 4](#_Toc62058007)

[14. Что такое динамический массив и для чего они применяются? 4](#_Toc62058008)

[15. Как создать динамический массив? 4](#_Toc62058009)

# 1. Что такое указатель?

Указатель – это обьект, который содержит в себе адрес других обьектов  (переменных, констант, указателей) или функций в памяти.

# 2. Как определить указатель?

Для определения указателя надо указать тип объекта, на который указывает указатель, и символ звездочки \*. Например, определим указатель на объект типа int:

|  |  |
| --- | --- |
| 1 | int \*p; |

Пока указатель не ссылается ни на какой объект. При этом в отличие от ссылки указатель необязательно инициализировать каким-либо значением. Теперь присвоим указателю адрес переменной:

|  |  |
| --- | --- |
| 1  2  3 | int x = 10;     // определяем переменную  int \*p;         // определяем указатель  p = &x;         // указатель получает адрес переменной |

Чтобы получить адрес переменной, нужно применить амперсанд (значок & (значок и, and))

Чтобы получить значение переменной, на которую ссылается указатель, нужно применить \* (звёздочку).

# 3. Как проинициализировать указатель?

Присвоить указателю адрес существующего обьекта.

# 4. Какие способы инициализации указателей вы знаете?

1. Присваивание указателю адреса существующего объекта:

• с помощью операции получения адреса:

int а = 5; // целая переменная

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

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

• с помощью значения другого инициализированного указателя:

int\* r = р;

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

int b[10]; // массив

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

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

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

pf = f; // присваивание адреса функции

2. Присваивание указателю адреса области памяти в явном виде.

char\* cp = (char \*) 0хВ8000000;

3. Присваивание пустого значения:

int\* suxx = NULL;

int\* rulez = 0;

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

• с помощью операции new: int\* n = new int; // 1 int\* m = new int (10); // 2 int\* q = new int [10]; // 3

• с помощью функции mallос: int\* u = (int \*)malloc(sizeof(int)); // 4

# 5. Как обратиться к переменной, адресуемой указателем?

• с помощью операции получения адреса:

int а = 5; // целая переменная

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

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

• с помощью значения другого инициализированного указателя:

int\* r = р;

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

int b[10]; // массив

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

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

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

pf = f; // присваивание адреса функции

# 6. Как получить адрес объекта?

**С помощью амерсанда(&)**

# 7. Какие операции можно выполнять над указателями?

**Операция разыменования предназначена для доступа к величине, адрес которой хранится в указателе. Эту операцию можно использовать как для получения, так и для изменения значения величины.**

char а; // переменная типа

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

\*р = 'Ю'; // присваивание значения динамической переменной // копирование содержимого ячейки памяти на которую указывает p в переменную a

а = \*р;

**Арифметические операции с указателями (сложение с константой, вычитание, инкремент и декремент) автоматически учитывают размер типа величин, адресуемых указателями. Эти операции применимы только к указателям одного типа и имеют смысл в основном при работе со структурами данных, последовательно размещенными в памяти, например, с массивами. Инкремент перемещает указатель к следующему элементу массива, декремент – к предыдущему. Фактически значение указателя изменяется на величину sizeof(тип). Если указатель на определенный тип увеличивается или уменьшается на константу, его значение изменяется на величину этой константы, умноженную на размер объекта данного типа**

**Разность двух указателей – это разность их значений, деленная на размер типа в байтах (в применении к массивам разность указателей, например, на третий и шестой элементы равна 3). Суммирование двух указателей не допускается. При записи выражений с указателями следует обращать внимание на приоритеты операций. В качестве примера рассмотрим последовательность действий, заданную в операторе.**

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

# 8. Что такое ссылка?

**Это синоним имени, указанного при инициализации ссылки. Это указатель, который всегда разыменовывается.**

# 9. Как объявить ссылку?

Формат объявления ссылки: тип & имя (тип амперсант имя) ; где тип — это тип величины, на которую указывает ссылка, & — оператор ссылки, означающий, что следующее за ним имя является именем переменной ссылочного типа, например: int kol; int& раl = kol: // ссылка раl — альтернативное имя для kol

# 10. Как создать переменную в динамической памяти?

Для динамического выделения памяти одной переменной используется **оператор** **new**:

New int;

Для доступа к выделенной памяти создается указатель:

int \*ptr = new int;

**Статическое выделение памяти** выполняется для [**статических**](https://ravesli.com/urok-51-staticheskie-peremennye/) и [**глобальных**](https://ravesli.com/urok-49-globalnye-peremennye/) переменных. Память выделяется один раз (при запуске программы) и сохраняется на протяжении работы всей программы.

**Автоматическое выделение памяти** выполняется для [**параметров функции**](https://ravesli.com/urok-13-parametry-i-argumenty-funktsij/) и [**локальных переменных**](https://ravesli.com/urok-48-lokalnye-peremennye-oblast-vidimosti-i-vremya-zhizni/). Память выделяется при входе в блок, в котором находятся эти переменные, и удаляется при выходе из него.

**Динамическое выделение памяти** — это способ запроса памяти из операционной системы запущенными программами по мере необходимости. Эта память не выделяется из ограниченной памяти стека программы, а выделяется из гораздо большего хранилища, управляемого операционной системой — **кучи**. На современных компьютерах размер кучи может составлять гигабайты памяти.

Когда вы динамически выделяете память, то вы просите операционную систему зарезервировать часть этой памяти для использования вашей программой. Если ОС может выполнить этот запрос, то возвращается адрес этой памяти обратно в вашу программу. С этого момента и в дальнейшем ваша программа сможет использовать эту память, как только пожелает. Когда вы уже выполнили с этой памятью всё, что было необходимо, то её нужно вернуть обратно в операционную систему, для распределения между другими запросами.

В отличие от статического или автоматического выделения памяти, программа самостоятельно отвечает за запрос и обратный возврат динамически выделенной памяти.

# 11. Как удалить переменную из динамической памяти?

Когда уже всё, что требовалось, выполнено с динамически выделенной переменной — нужно явно указать для С++ освободить эту память. Для переменных это выполняется с помощью **оператора** **delete**:

*// Предположим, что ptr ранее уже был выделен с помощью оператора new*

delete ptr; // возвращаем память, на которую указывал ptr, обратно в операционную систему

ptr = 0; // делаем ptr нулевым указателем (используйте nullptr вместо 0 в C++11)

# 12. На что следует обращать особое внимание при работе с динамической памятью?

Нужно освобождать память в конце.

* d**elete указатель;** — для одного элемента
* **delete[] указатель;** — для массивов

# 13. Какие способы обращения к элементам массива вы знаете?

1) С использованием квадратных скобок

| **C++**[Выделить код](https://www.cyberforum.ru/) | |
| --- | --- |
| |  |  | | --- | --- | | 1 | array[i] | |  |

2)При помощи указателей

| **C++**[Выделить код](https://www.cyberforum.ru/) | |
| --- | --- |
| |  |  | | --- | --- | | 1 | \*(array+i) | |  |

# 14. Что такое динамический массив и для чего они применяются?

**Динамическим** называется [массив](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B8%D0%B2_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), размер которого может изменяться во время исполнения программы. Возможность изменения размера отличает динамический массив от статического, размер которого задаётся на момент компиляции программы. Для изменения размера динамического массива [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), поддерживающий такие массивы, должен предоставлять встроенную функцию или оператор. Динамические массивы дают возможность более гибкой работы с данными, так как позволяют не прогнозировать хранимые объёмы данных, а регулировать размер массива в соответствии с реально необходимыми объёмами.

Применяются, когда неизвестна точная размерность массива.

# 15. Как создать динамический массив?

1. Запрашиваем размер массива [num]

2. Выделяем память для массива

New int[num]

Или int \*p\_darr=new int[num] с помощью указателя.

3. По завершению работы программы, очищается память с помощью delete