6. Указатели,константные указатели, арифметика указателей

Программирование и алгоритмизация

Практические занятия

БИВТ-24-17

Надежда Анисимова

ms teams m2102039@edu.misis.ru

Проверка себя

1. Какие есть массивы в c++?
2. В чём разница между size и capacity?
3. Особенности c-строк и std::string?
4. По какому принципу работает очередь?

Указатели

01

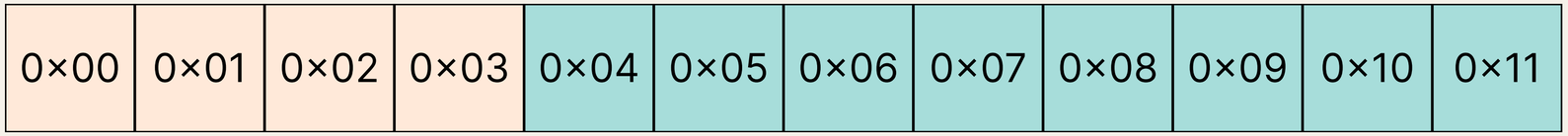
Напоминалка

- Переменная - именованное хранилище, именованный участок памяти под определенный тип

Указатель – переменная, значением которой является адрес ячейки памяти

Указатель типа **T*** хранит адрес объекта типа **T**

int value; **int* ptr;**



Обе переменные лежат в памяти стека

nullptr

ключевое слово для **безопасного** обозначение нулевого указателя

```
int* ptr = nullptr;
```
нулевой указатель

! в памяти не обязательно представлен как 0
используется для обозначения **недоступного адреса**

Как и любые другие переменные, указатели рекомендуется сразу инициализировать!

Символ * - модификатор типа

```
int*  p;      int  *p;
```

допустимы пробелы до и после *, обе записи обозначают, что p - указатель на тип int

Однако, если объявлять несколько указателей, к каждой переменной нужно добавить модификатор

```
int  *p1, *p2;
```

Обе переменные - указатели

```
int* p1, p2;
int  *p1, p2;
```

p1 - указатель, p2 - переменная типа int

Оператор обращения к адресу

Если использовать **& как унарный оператор** , то можно получить адрес объекта в памяти

Так как указатели используются для хранения адресов, то получив адрес через & , его **можно присвоить указателю**

```
int value = 4;
```

```
int* ptr = &value;
```

Стек

0×7ffc21a79a9c	байт 1
0×7ffc21a79a9d	байт 2
0×7ffc21a79a9e	байт 3
0×7ffc21a79a9f	байт 4
0×7ffc21a79aa0	байт 5
0×7ffc21a79aa1	байт 6
0×7ffc21a79aa2	байт 7
0×7ffc21a79aa3	байт 8
0×7ffc21a79aa4	байт 9
0×7ffc21a79aa5	байт 10
0×7ffc21a79aa6	байт 11
0×7ffc21a79aa7	байт 12
0×7ffc21a79aa8	байт 13
0×7ffc21a79aa9	байт 14

Здесь хранится значение 4

int value = 4;

Здесь хранится адрес переменной value, то есть **0×7ffc21a79a9c**

int* ptr = &value;

Оператор разыменования

Если использовать * **как унарный оператор** , то можно имея адрес в памяти, получить значение переменной, расположенной по этому адресу

```
int value = 4;
```

значение

4

```
int* ptr = &value;
```

адрес переменной value

0×7ffc21a79a9c

```
int newValue = *ptr;
```

значение

4

Операторы * и &

1. Модификатор типа

```
int* ptr = nullptr;
```
указатель

```
int& ref = value;
```
ссылка

2. Унарный оператор

```
value = *ptr;
```
оператор разыменования

```
ptr = &value;
```
оператор обращения к адресу

3. Бинарный оператор

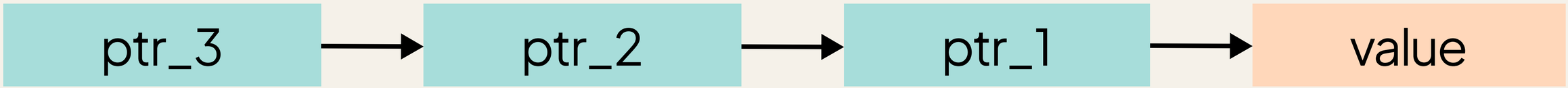
```
value = value * value;
```
оператор умножения

```
bitRes = value1 & value2;
```
оператор побитовое И

Указатели на указатели

Теоретически нет предела количеству модификаторов типа *

```
int value = 4;
int* ptr_1 = &value;
int** ptr_2 = &ptr_1;
int*** ptr_3 = &ptr_2;
```



Указатели vs ссылки

02

Напоминалка

- Ссылка - альтернативное имя переменной (объекта), псевдоним

Указатель - объект, может вести себя как любые другие объекты, например, как объекты типа int, double

Оба используются для косвенного доступа к другим объектам

```
int* ptr = &value;
```

```
int& ref = value;
```

Стек

0×7ffc21a79a9c	байт 1
0×7ffc21a79a9d	байт 2
0×7ffc21a79a9e	байт 3
0×7ffc21a79a9f	байт 4
0×7ffc21a79aa0	байт 5
0×7ffc21a79aa1	байт 6
0×7ffc21a79aa2	байт 7
0×7ffc21a79aa3	байт 8
0×7ffc21a79aa4	байт 9
0×7ffc21a79aa5	байт 10
0×7ffc21a79aa6	байт 11
0×7ffc21a79aa7	байт 12
0×7ffc21a79aa8	байт 13
0×7ffc21a79aa9	байт 14

value или ref – имена для этой области памяти

```
int value = 4;
```

```
int& ref = value;
```

```
int* ptr = &value;
```

```
int* ptr = &ref;
```

0×7ffc21a79a9c

- **Оба используются для косвенного доступа к другим объектам**

```
int* ptr = &value;
```

```
int& ref = value;
```

- **НО указатель - объект, а ссылка - другое имя**

  Указатели могут быть присвоены/скопированы

  Могут указывать сначала на один объект, потом на другой

  Могут быть не инициализированными

- **Можно определить ссылку на указатель - получится альтернативное имя для указателя**

```
int*& ref_to_ptr = ptr;
```

Указатели и спецификатор const

03

напоминалка

Ссылка на константу (reference to const)

Ссылка - альтернативное имя, поэтому сама по себе константной быть не может, но может ссылаться на объект, свойства которого станут константными

```
// ok
const int val = 50;
const int& ref = val;

int val = 50;
const int& ref = val;
```

```
// ошибка
const int val = 50;
int& ref = val;
```

Указатель на константу (pointer to const)

Аналогично ссылке, с помощью указателя на объект можно наложить дополнительное ограничение в виде константности, то есть запретить через указатель менять значение переменной

*ptr = 30; - запретить так менять значение

// ok

```
const int val = 50;
const int* ptr = &val;
```

```
int val = 50;
const int* ptr = &val;
```

// ошибка

```
const int val = 50;
int* ptr = &val;
```

Константный указатель (const pointer)

Но в отличие от ссылки, указатель - объект, поэтому можно так же запретить менять значение самого указателя, то есть запретить присваивать ему новый адрес

```
cptr = &value; - запретить так менять значение
```

```
int *const cptr = &value;
```

после * добавить const

А что получится есть объединить?

- Указатель на константу – const int* ptr
- Константный указатель – int *const ptr;

```
const int  *const ptr = &value;
```

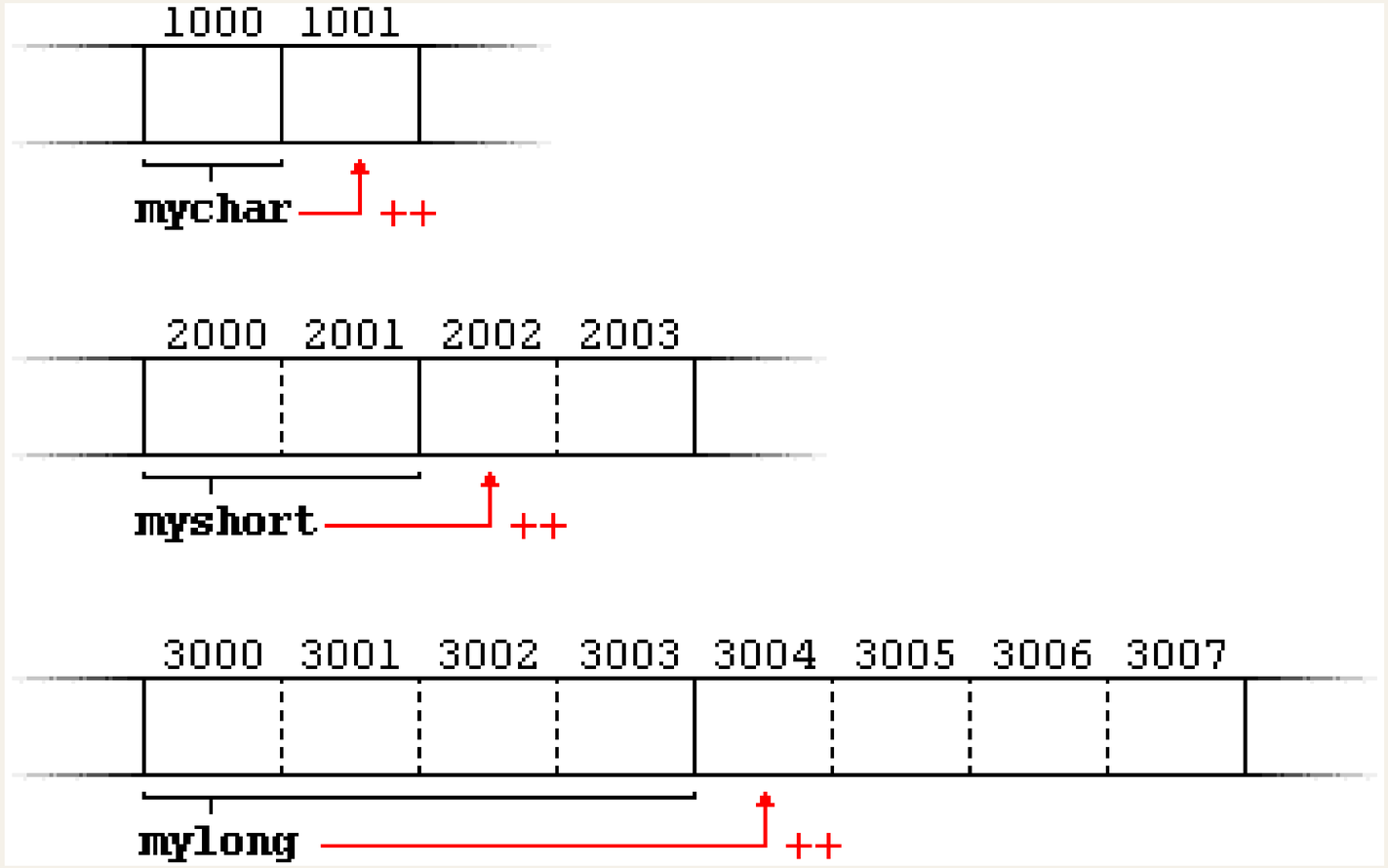
- Нельзя менять переменную: *ptr = newValue;
- Нельзя менять адрес: ptr = &newValue;

Арифметика указателей

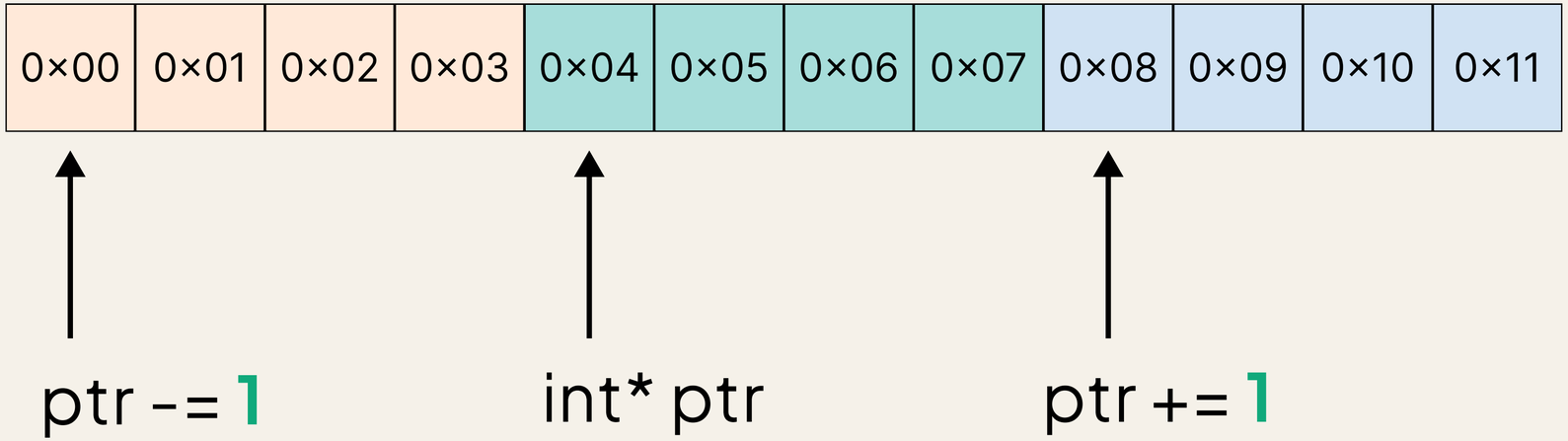
04

Для указателей определены только операции **сложения и вычитания**

НО, при увеличении указателя на 1, он будет указывать на следующий элемент того же типа



Указатель + N <=> Адрес + N * sizeof(тип)



На практике арифметика указателей имеет смысл, если указатель на массив, то есть если вы точно знаете, что хранится в последующем/предыдущем участке памяти

Домашнее задание

Закончить текущий контест