Лекция 3 – Указатели, массивы, ссылки и вектора

Воронин Андрей Андреевич

Кафедра прикладной математики и информатики

7 октября 2019 г.

Введение

Определение

Массив – непрерывно расположенная в памяти последовательность элементов одного типа.

Массив из 6 элементов типа **char**:

```
char v[6];
```

Указатель на элемент типа **char**:

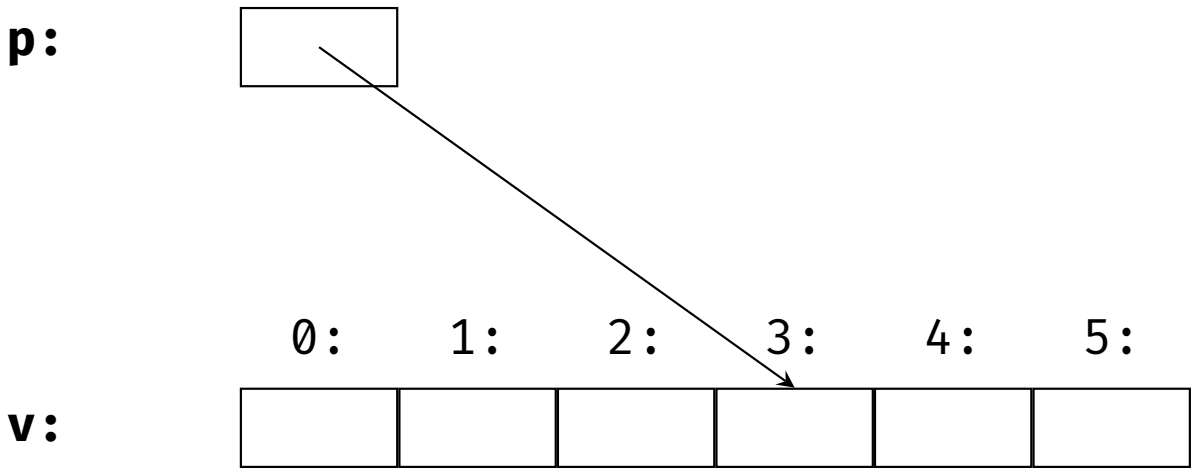
```
char* p;
```

Взятие ссылки и разыменовывание указателя:

```
char *p = &v[3]; // p указывает на 4 элемент массива
char x = *p; // разыменование указателя -- получаем
 ↪  значение на которое указывает указатель
```

Префиксный оператор & является оператором взятия ссылки.

Указатель и массив



Пример 1

```
void copy_for_some_purpose()
{
    int v1[10] = {0,1,2,3,4,5,6,7,8,9};
    int v2[10];

    for(auto i=10; i!=10; ++i)
        v2[i] = v1[i];
    // ...
}
```

Пример 2

```
void print()
{
    int v[] = {0,1,2,3,4,5,6,7,8,9};

    for (auto x : v)
        cout << x << '\n';

    for (auto x : {12,73,08,23,22})
        cout<< x << '\n';
}
```

Пример 3

```
void increment()
{
    int v[] = {0,1,2,3,4,5,6,7,8,9};

    for (auto &x : v)
        ++x;
}
```

Ссылки

Во время объявления переменной префикс & означает ”сыллка на”.
Ссылка похожа на указатель, за тем исключением, что нет необходимости разыменовывать указатель для доступа к значению.
Также нельзя сменить объект на который она указывает.
Ссылки в первую очередь важны для определения аргументов функции:

```
void sort(vector<double>& v); // сортируем вектор v (v
 ↪  это вектор значений double)
```

Объявляя v как ссылку мы изменяем поведение функции, запрещая копирование аргументов.

Константная ссылка кроме того, гарантирует неизменяемость аргументов внутри функции.

```
double sum(const vector<double> &v);
```

Нулевой указатель

```
double *pd = nullptr;
Link<Record> *lst = nullptr; // указатель на
 ↪  параметризуемый контейнер
int x = nullptr;                          // error:
 ↪  nullptr is a pointer not an integer
```

В более раннем коде можно встретить использование `0` или `NULL` вместо использования **`nullptr`**. Тем не менее использование **`nullptr`** избавляет от коллизии с целочисленными литералами и указателями.

Нулевой указатель

Часто важно проверять аргументы функции на тот случай, что они на самом деле указывают на существующий объект.

```
int count_x (const char *p, char x)
{
    if (p==nullptr)
        return 0;
    int count = 0;
    for (; *p!=0; ++p)
        if (*p==x)
            ++count;
    return count;
}
```

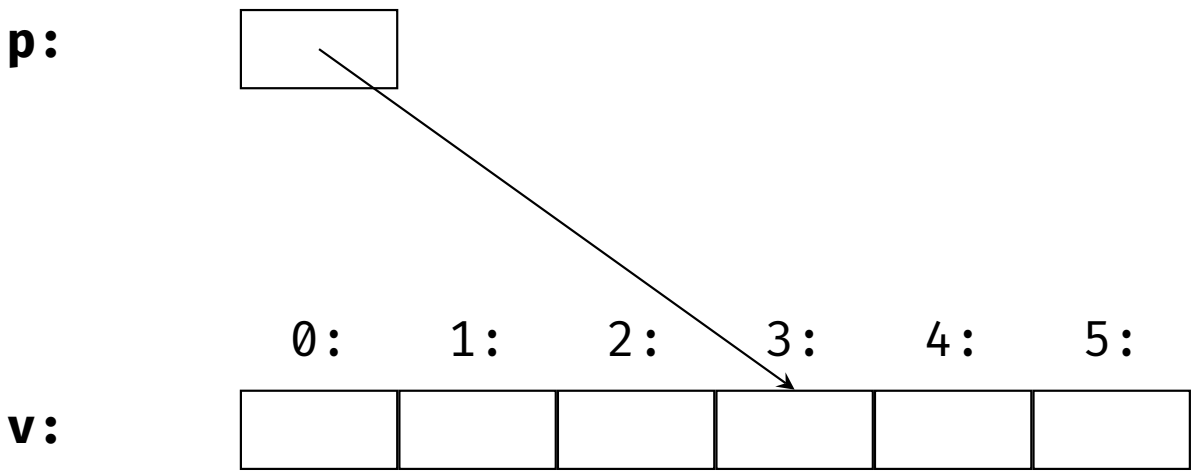
Выше мы предполагаем, что `char *` это так называемая строка в C-стиле, которая оканчивается нулевым терминирующим символом `'\0'`.

Нулевой указатель

В предыдущем примере мы не использовали инициализирующее значение в цикле **for**, мы можем использовать цикл **while**

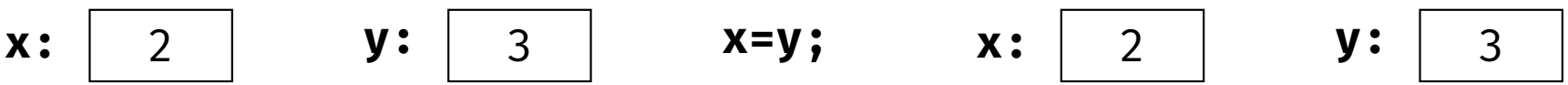
```
int count_x (const char *p, char x)
{
    if (p==nullptr)
    return 0;
    int count = 0;
    while (*p)
    {
        if (*p==x)
            ++count;
        ++p;
    }
    return count;
}
```

Использование аппаратных возможностей



Присваивание переменных

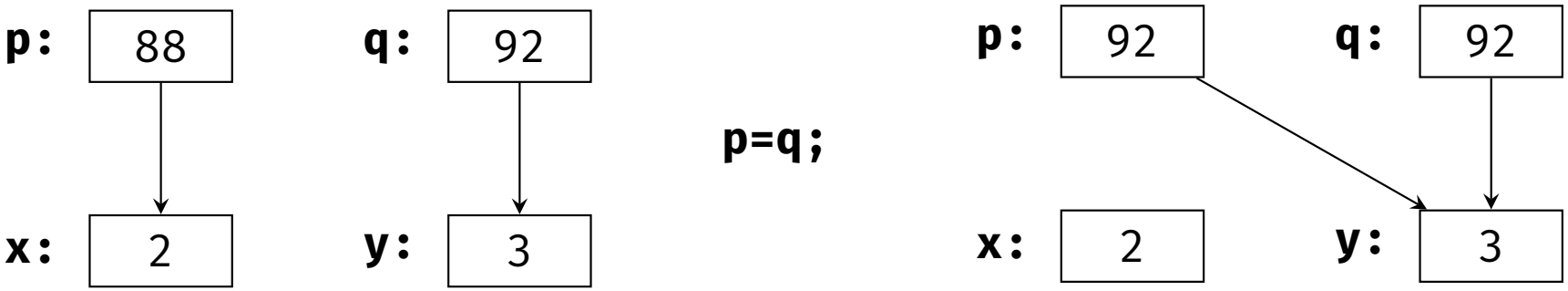
```
int x = 2;
int y = 3;
x = y; // x становится равным 3
// Note: x==y
```



если после этого `x=99;`, то из-за этого значение y останется равным 3.

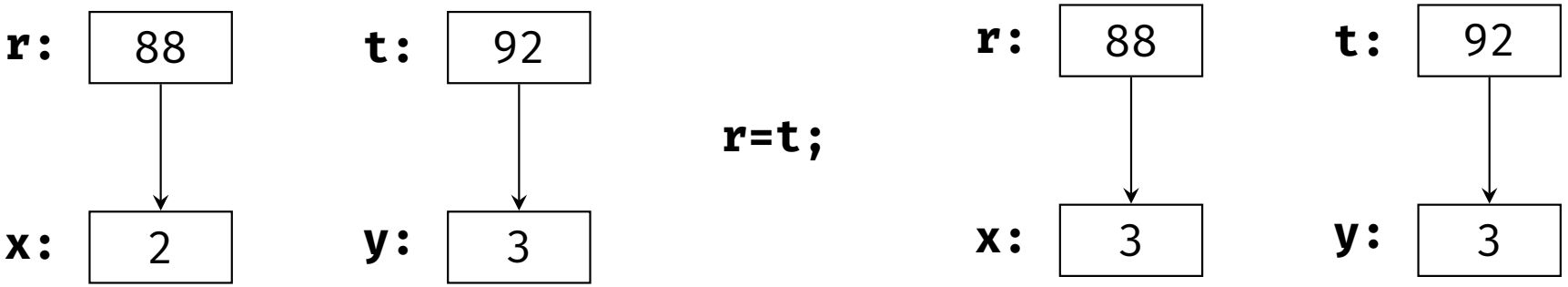
Указатели

```
int x = 2;
int y = 3;
int *p = &x;
int *y = &y; // сейчас p != q и *p != *q
p = q;       // p становится равным &y; сейчас p == q
 ↪  и *p == *q
```



Ссылки

```
int x = 2;
int y = 3;
int &r = x; // r ссылается на x
int &t = y; // t ссылается на y
r = t;      // читаем по ссылке t и записываем по
 ↪  ссылке r
```



Инициализация

```
int x = 7;
int &r {x}; // связываем r с x (r ссылается на x)
r = 8;      // присваеваем значение к чему-то на что
 ↪  ссылается r

int &r2;    // error: неинициализированная ссылка
r2 = 99;    // присваеваем значение к чему-то на что
 ↪  ссылается r2
```

Инициализация

```cpp
#include <vector>
std::vector<int> myVector; // создаем пустой вектор
 ↪  типа int
myVector.reserve(10);      // резервируем память под
 ↪  10 элементов типа int

int myArray[10]; // аналог в виде массива
```

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> myVector(10);   // объявляем вектор
     ↪  размером в 10 элементов и инициализируем их
     ↪  нулями
    // вывод элементов вектора на экран
    for(int i = 0; i < myVector.size(); i++)
        cout << myVector[i] << ' ';
    return 0;
}
```

Инициализация

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> vec1(3);
    // инициализируем элементы вектора vec1
    vec1[0] = 4;
    vec1[1] = 2;
    vec1[2] = 1;
    vector<int> vec2(3);
    // инициализируем элементы вектора vec2
    vec2[2] = 1;
    vec2[1] = 2;
    vec2[0] = 4;
    // сравниваем массивы
    if (vec1 == vec2) {
        cout << "vec1 == vec2" << endl;
    }
    return 0;
}
```

Вектор

```
#include <iostream>
#include <vector>
#include <iterator> // заголовочный файл итераторов
using namespace std;

int main()
{
    vector<int> vec1; // создаем пустой вектор
    // добавляем в конец вектора vec1 элементы 4, 3, 1
    vec1.insert(vec1.end(), 4);
    vec1.insert(vec1.end(), 3);
    vec1.insert(vec1.end(), 1);
    // вывод на экран элементов вектора
    copy( vec1.begin(),   // итератор начала массива
          vec1.end(),     // итератор конца массива
          ostream_iterator<int>(cout," ")   //итератор
          ↪  потока вывода
    );
    return 0;
}
```

Советы I

1. Не паникуй!© Все станет яснее со временем.
2. Не используй только встроенные фичи в язык – кроме этого еще есть стандартная библиотека.
3. Нет необходимости знать все тонкости языка для того чтобы писать хорошие программы.
4. ”Пакуй” целостные группы операций в функции.
5. Функция должна отвечать за одну логическую операцию.
6. Старайся создавать функции короткими.
7. Используй перегрузку когда функции производят одну и ту же операцию над различными типами.
8. Если функция может быть рассчитана на этапе компиляции, то используй **`constexpr`**.
9. Необходимо понимать как языковые приметивы отображаются на аппаратное обеспечение.
10. 300000000 хуже чем 300’000’000.
11. Избегай сложных выражений.
12. Избегай преобразований с округлением.

Советы II

13. Минимизируй область видимости переменной.
14. Избегай ”магических” констант.
15. Предпочитай неизменяемые данные.
16. Объявляй только одно имя переменной в строчке.
17. Локальные переменные и распространенные переменные должны иметь короткие имена, а необычные переменные объявленные далеко от вычислений должны иметь длинные имена.
18. Избегай имен похожих друг на друга.
19. Избегай имен вида `ALL_CAPS`.
20. Предпочитай инициализацию в виде `int x {7.9};`
21. Используй **`auto`** чтобы избегать повторения имен типов.
22. Избегай неинициализированных переменных.
23. Не объявляй переменную до того как можешь ее инициализировать.
24. **`unsigned`** типы используются только для битовых операций.
25. Используй указатели просто и прямолинейно.
26. Не пиши комментариев смысл которых понятен из кода.
27. Используй **`nullptr`** вместо `0` или `NULL`.
28. Объясняй свои намерения в комментариях.
29. Поддерживай однородную табуляцию.

Литературные источники I

Stroustrup B. — A Tour of C++ Second Edition. — Addison-Wesley, 2018. — (C++ in-Depth Series). — ISBN 9780134997834. — URL: https://books.google.ru/books?id=UGtRtAEACAAJ.

C++ Core Guidelines. /. — Под ред. B. Stroustrup, H. Sutter. — 16.06.2019. — URL: http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines.

Google C++ Style Guide. /. — Google. — URL: https://google.github.io/styleguide/cppguide.html.

Справка по C++. — URL: https://ru.cppreference.com/w/.