HIGHER SCHOOL OF ECONOMICS
NATIONAL RESEARCH UNIVERSITY

Introduction to Programming

References, Consts and Structures

Sergey Shershakov

#6/29 Jan 2019

Test 5 (4 pts)

https://goo.gl/forms/jI7OHXivvuMmgqYh1

CV-QUALIFIERS

`const` and `volatile`

- CV(-modifiers) stands for `const` and `volatile`

- `const`
  - an object (or memory) is initialized once and cannot be changed further

- `volatile`
  - states that an object can be changed somewhere even if you do not provide a modification statement for the object
  - rarely used, for the sake of optimization performed by the compiler

const Types and Objects

```
int main()
{
    int a = 15;

    const int b = 10;
    //b = 17;                    // can't modify a const object

    int const b1 = 10;

    return 0;
}
```

according to the codestyle

the same

correct, but we won't use such a notation!

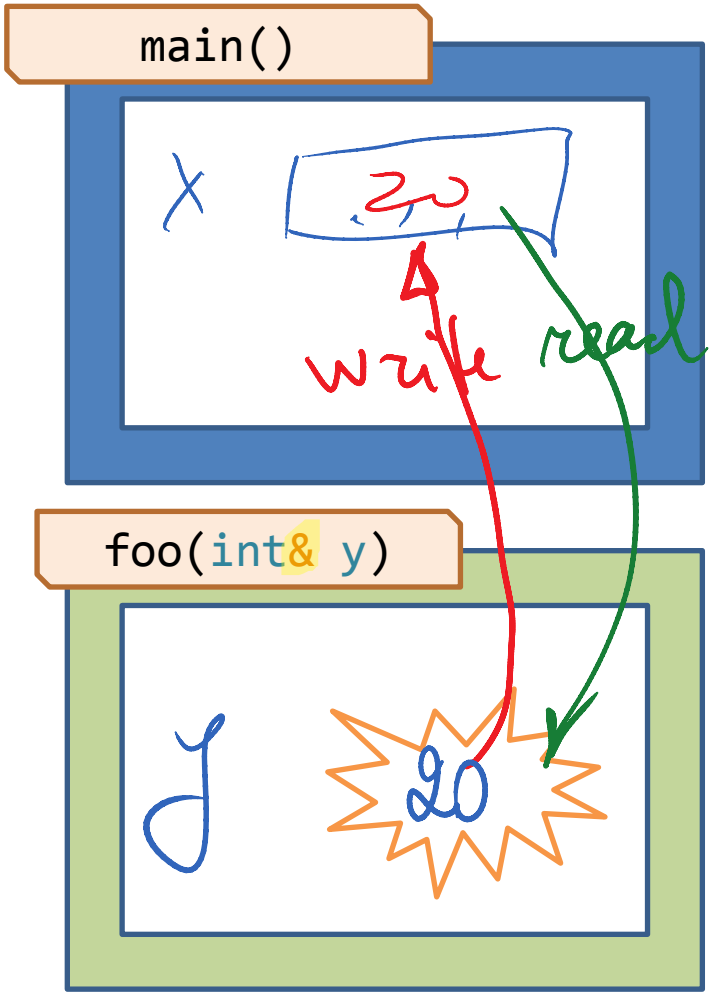
Reference

Mole hole, wormhole

- Reference is another name for an object.
- Reference type:

  `typename&`

```
int x = 10;
cout << x;     // 10

int& x1 = x;   // binding
cout << x1;    // 10


x1 = 15;
cout << x1;    // 15
cout << x;     // 15


x = 42;
cout << x;     // 42
cout << x1;    // 42
```

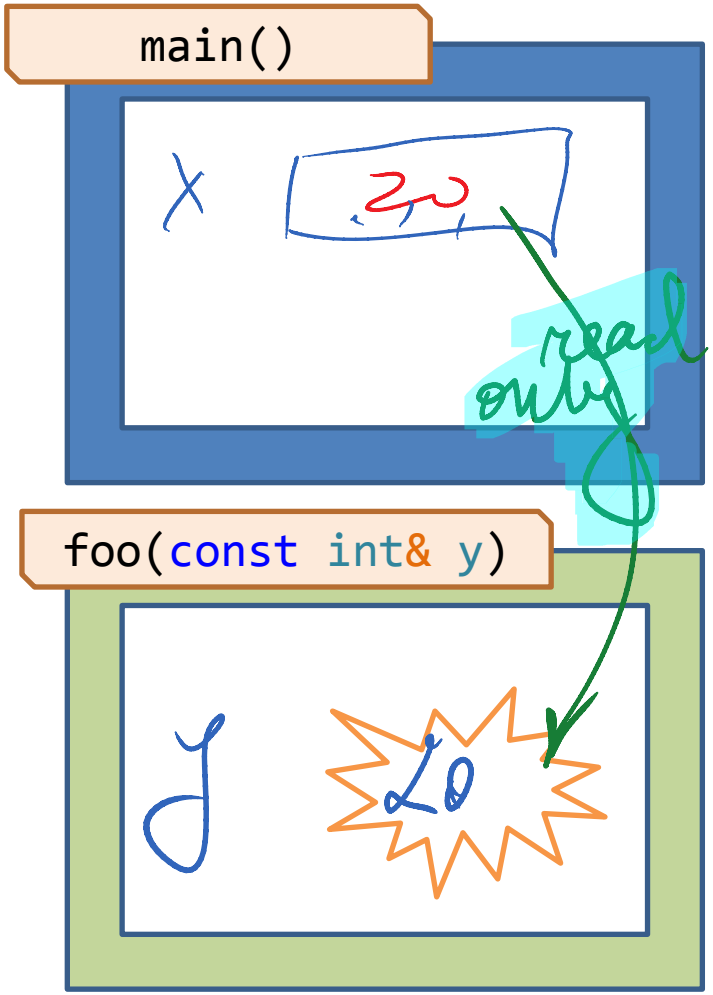


`const` Reference

- Reference is another name for an object.
- Reference type:

  `const typename&`

```
int x = 10;    // x == 10

int& x1 = x;   // binding
x1 = 15;

cout << x1;    // 15
cout << x;     // 15

const int& x2 = x; // binding const

//x2 = 20;       // compilation error!

cout << x2;    // 15
```

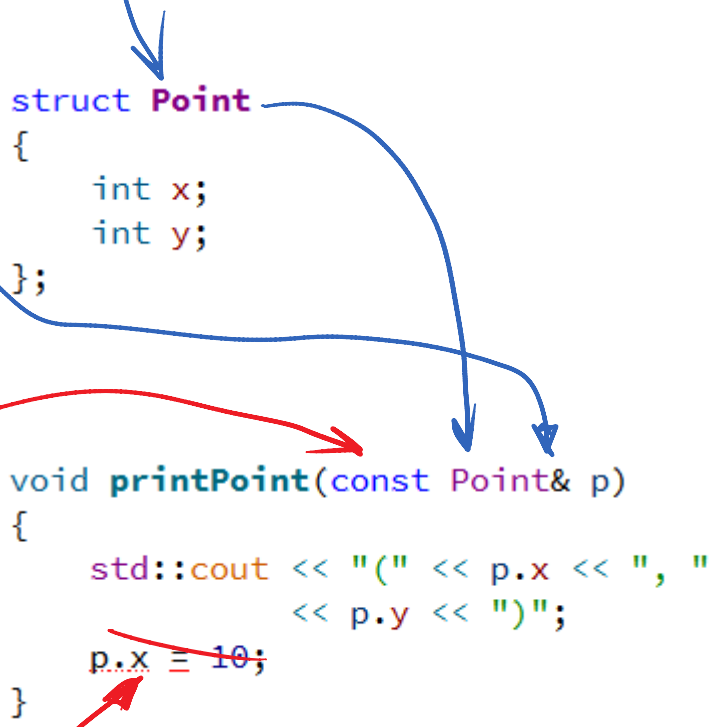


Thumb-Rules on Using *Refs* and *Const Refs*

1. If a method has a parameter represented by a *complex object* (e.g., any one bigger than any POD such `int`, `double`, `T*`), and a method does not need a copy of an object represented by the parameter, it is better to pass the parameter by using *reference* or *const reference*.
   - POD types are more efficient to be passed *by value*.

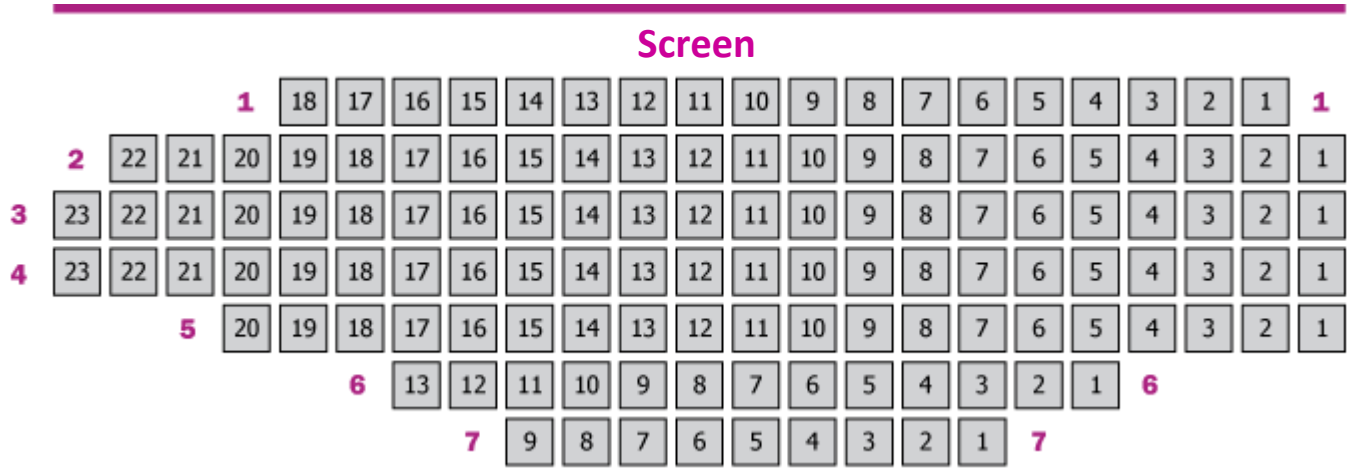
2. Let a method have a parameter given *by reference*, and let the method not change the value of the parameter, then the parameter must be given by using a **const** ref.

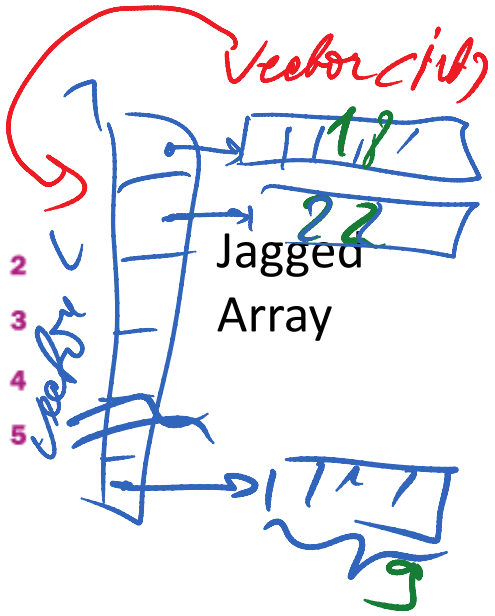
3. If an object (e.g., a parameter) of a method is a *const* (*ref*), all derivatives of the object (its member fields, member methods and so on.) are also *const*.



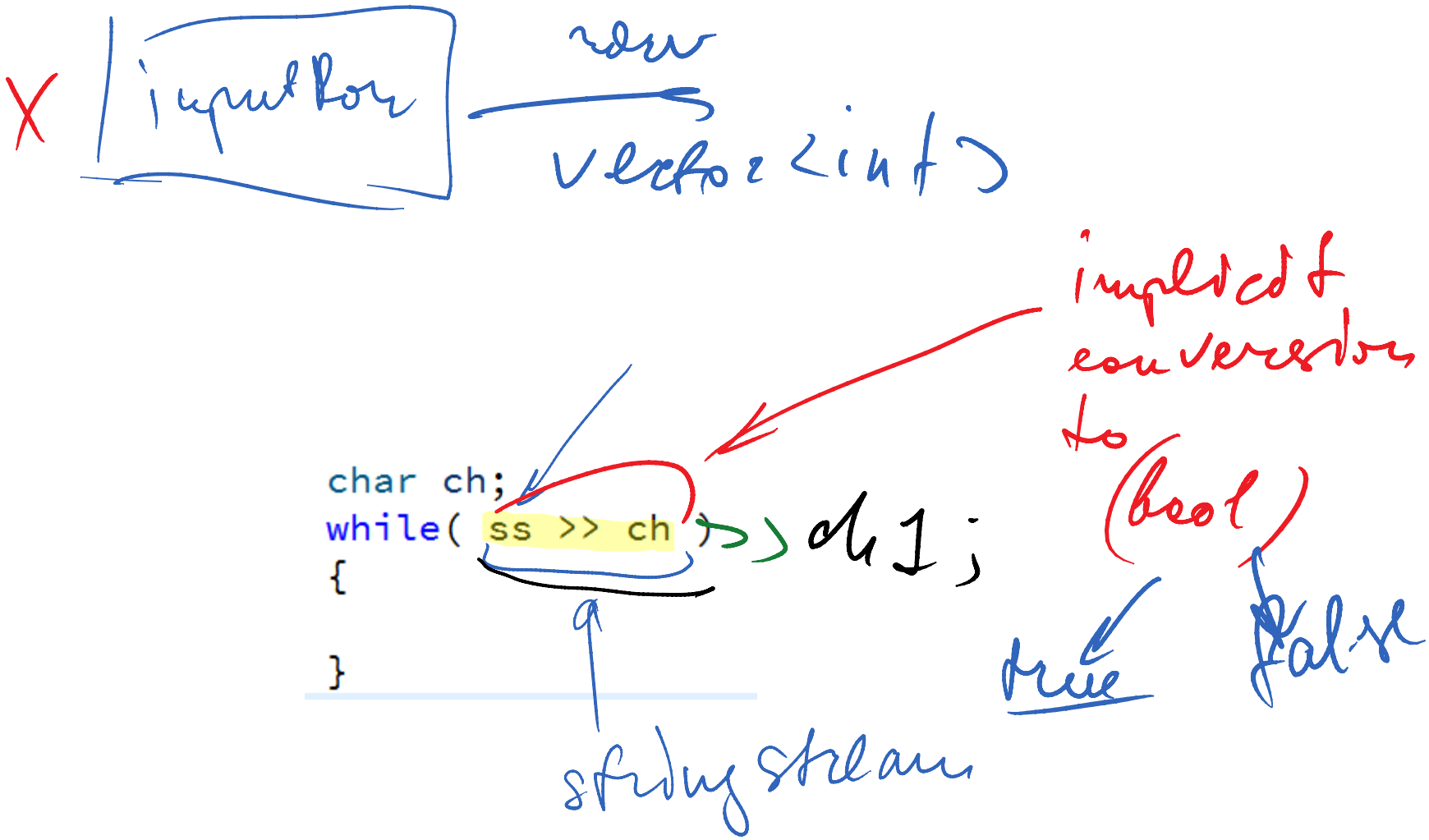
Contest 2 Problem 16

Let's Go to a Cinema!



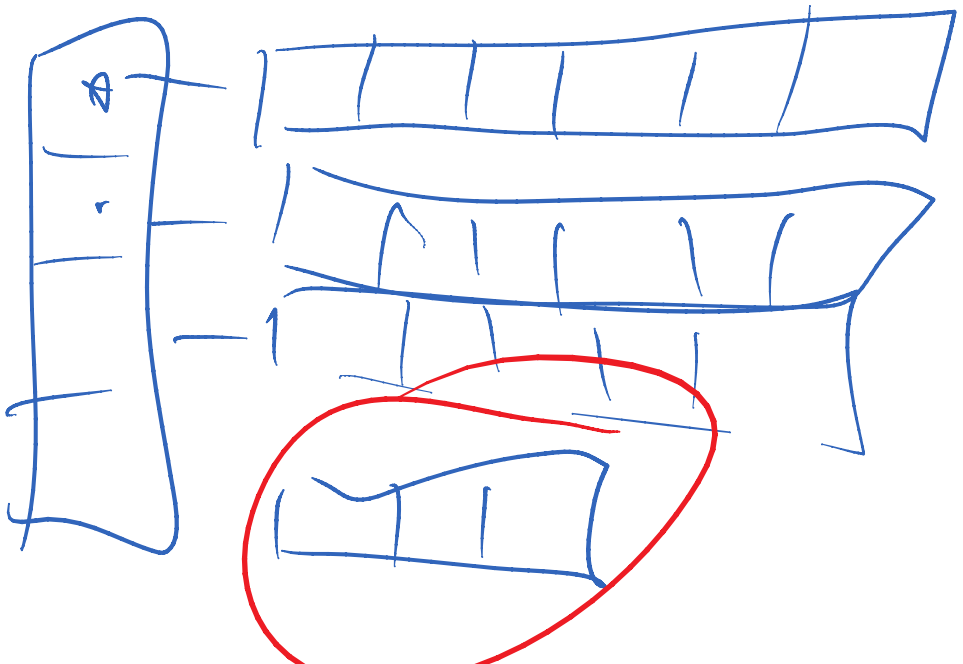


1) input data: m rows, n_i seats for each i-th row; 1 — the seat is sold, 0 — the seat is free;
2) print data in a different format: a row per line, * is for sold seats, . is for free; sold/total ratio in the end of each row/line;
3) someone would like to buy k adjacent seats in the same row; one needs to determine whether it is possible or not;
4) how to modify the printing method for highlighting the free k seats by using "XXXX" notation?


inputFor
row
vector<int>
implicit conversion to (bool)
char ch;
while(ss >> ch)
{
}
ch1;
true
false
string stream

Defining a Matrix as a Vector of Vectors

```
#include <vector>

typedef std::vector< std::vector<int> > IntMatrix;

IntMatrix m1;


// represents individual rows
typedef std::vector<std::string> IntVector;
```



STRUCTURES

Structure as a Compound Type

the `struct` keyword

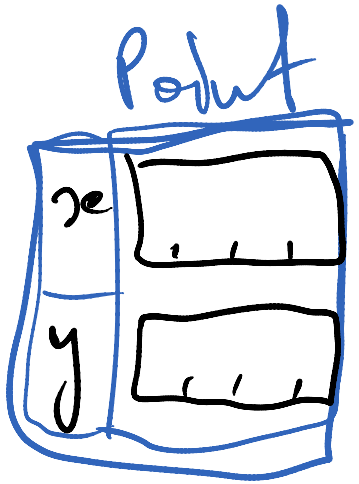
the name for a new (custom) type

```
struct Point
{
    int x;
    int y;
};
```

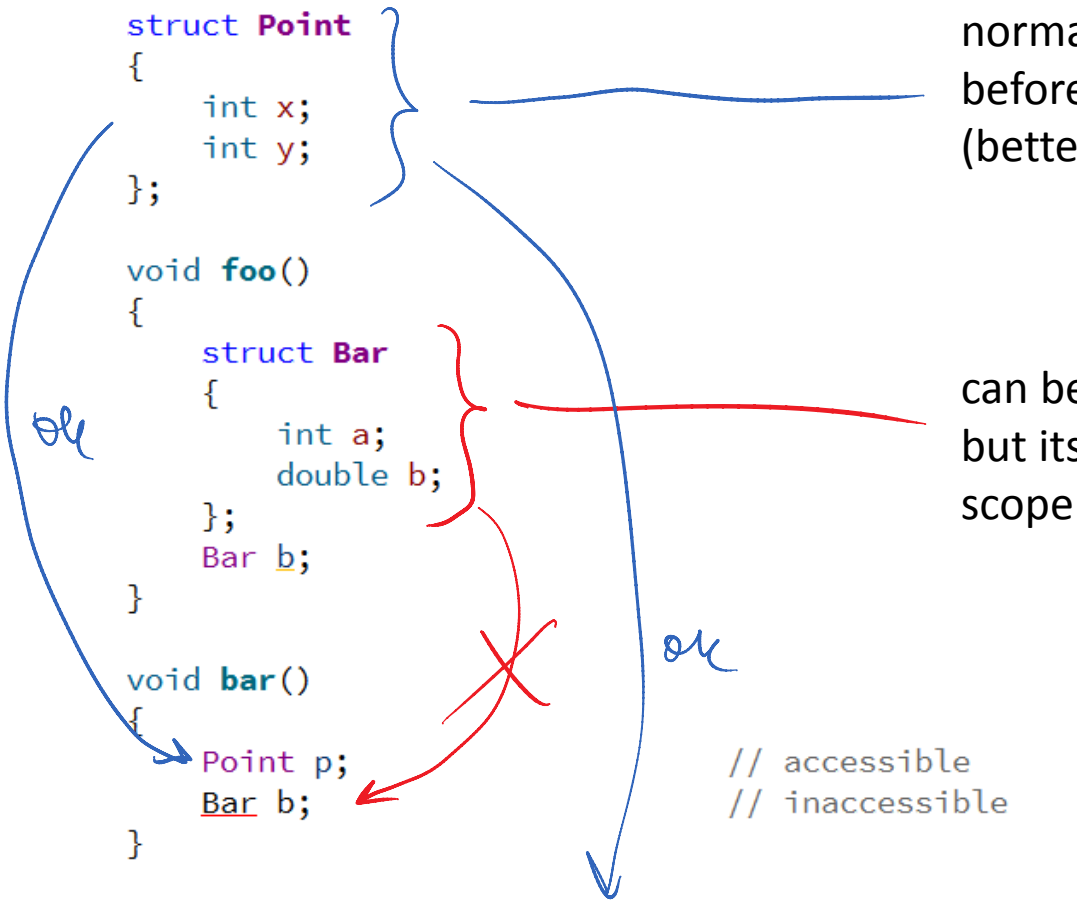
opening and closing braces

structure *members* (*fieds*)

terminates the structure declaration



Where to Define a Structure?



```
struct Point
{
    int x;
    int y;
};

void foo()
{
    struct Bar
    {
        int a;
        double b;
    };
    Bar b;
}

void bar()
{
    Point p;          // accessible
    Bar b;            // inaccessible
}

void printPoint(const Point& p)
{
    std::cout << "(" << p.x << ", " << p.y << ")";
    p.x = 10;
}
```

normally to be put somewhere
before the first use
(better to use a separate header (.h)

can be declared inside a function
but its visibility is restricted by the
scope of the function

Initialization of a Structure

```
Point p;                // default initialization (random values for x and y)
Point p1 = { 10, 15 };  // init by a list of values (x = 10, y = 15 )
Point p2 { 20, 25 };    // the same, new syntax as of C++11
Point p3 = p1;          // init p3 by the value of an object p1 of the same type
Point p4;
p4 = p1;                // re-assign the value of p4 by p1 point
```