Exercise Session

Week 11

Adel Gavranović
agavranovic@student.ethz.ch

Overview

- polybox for session material
- mail to TA

Today’s Topics

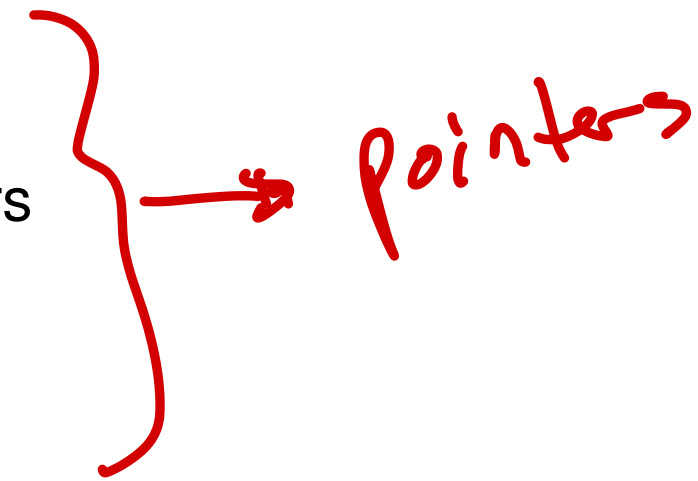
Introduction

Meanings of & and ∗

References vs Pointers

Pointer Arithmetic

Exercise: ”Push Back”



Introduction

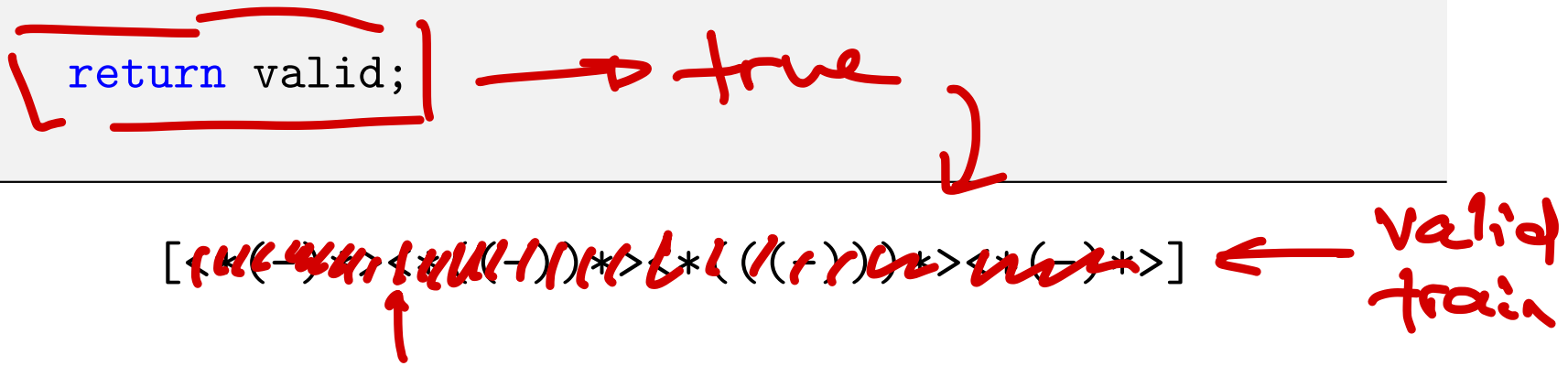
- One of the current tasks is running the newest version of the autograder, so if you find any bugs (or typos) send me an email

Comments on last [code]expert Exercises

- use more comments and try to format them well (don’t get too slacky now!)
- Exercise ”Trains”: many had this one function wrong, so I’m going to cover it here

EBNF Exercise ”Trains”

```
// composition = "<" open loco ">"
// compositions = composition | { composition } .
bool compositions(std::istream& is) {
    bool valid = composition(is);
    while (valid && lookahead(is) == '<') {
        valid = valid && composition(is);
    }

    return valid;
}
```

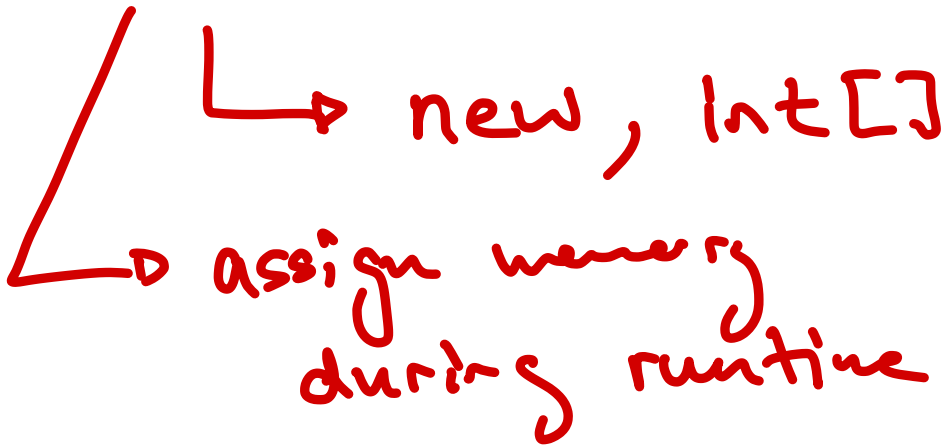


Questions or Comments re: Exercises?

Learning Objectives Checklist

Now I...

- ☐ can explain the differnce between a reference and a pointer
- ☐ can trace programs that use pointers and pointer arithmetic
- ☐ can write programs that use pointers and pointer arithmetic
- ☐ can trace programs that use dynamic memory
- ☐ can write programs that use dynamic memory



Questions?

Meanings of &

The symbol `&` can disorient many people approaching `C++` .
It is important to realize that this symbol has *3 different meanings*, depending on its position in the code:

Meanings of &

1. the bitwise AND operator
   ```
   z = x & y;
   ```
2. to *declare* a variable as a reference
   ```
   int& y = x;
   ```
3. to *take the address* of a variable (address operator)
   ```
   int *ptr_a = &a;
   ```

Meanings of ∗

Same with the symbol ∗:

Meanings of ∗

1. the arithmetic multiplication operator
   ```
   z = x * y;
   ```
2. to *declare* a pointer variable
   ```
   int * ptr_a = &a;
   ```
3. to *take the content* of a varibale *via* its pointer (dereference operator)
   ```
   int a = *ptr_a;
   ```

accessing the actual value at that addr.

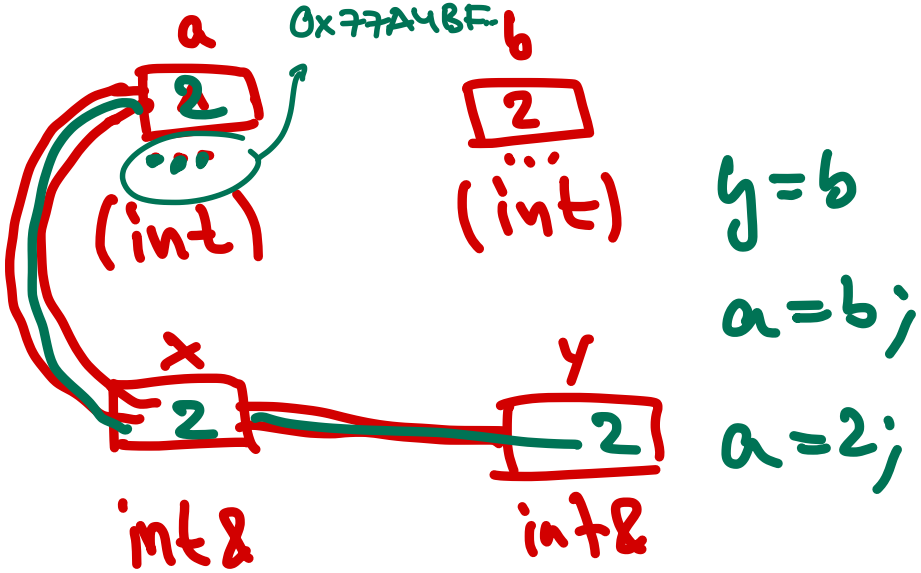
Questions?

References

```
void references(){
  int a = 1;
  int b = 2;
  int& x = a;
  int& y = x;
  y = b;

  std::cout
  << a << " "
  << b << " "
  << x << " "
  << y << std::endl;
}
```

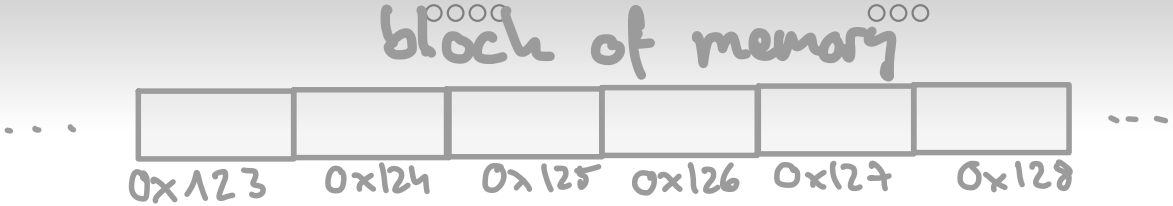
Task

Trace this program and write down the expected output



6 2

Pointers




```
void pointers(){
  int a = 1;
  int b = 2;
  int* x = &a;
  int* y = x;

  *y = b;

  std::cout
  << a << " "
  << b << " "
  << x << " "
  << y << std::endl;

  y = 0;
}
```

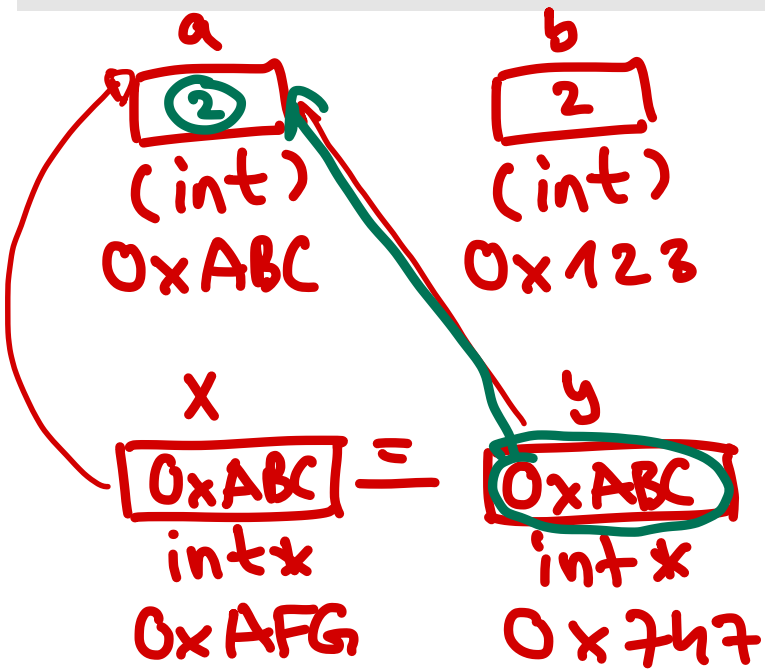
forgot to incl. this line

*y ~ deref. the pointer "y".

(y = 0; ← not imp.)

Task

Trace this program and write down the expected output



//output: 2 2 0xABC 0xABC

Pointers & Addresses

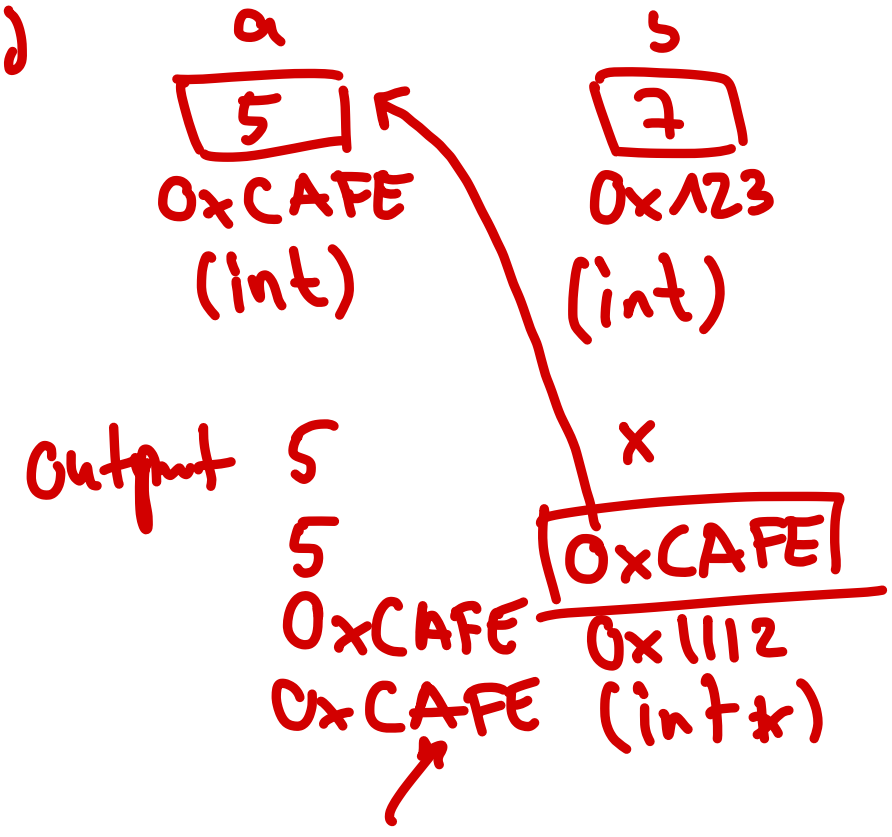
```
void ptrs_and_addresses(){
  int a = 5;
  int b = 7;

  int* x = nullptr;
  x = &a;

  std::cout << a << "\n";
  std::cout << *x << "\n";

  std::cout << x << "\n";
  std::cout << &a << "\n";

  x = &b;
  *x = 1;
}
```

"points at nothing"

not imp

Task

Trace this program and write down the expected output



Questions?

Bug hunt

Exercise

Find and fix (at least) 3 problems with the code in the code in Pointers_On_Arrays.pdf

Pointers and Arrays

Exercise

1. Trace the code in Reverse_Copy.pdf
2. determine a POST-condition for the function
   `f(int* b, int* e, int* o);`
3. Which inputs are valid? (see slides)
4. Make the function `const`-correct[1]

[1] If the whole `const*const&`-stuff confuses you, check out the summary for that topic on the course page.

Constness and Pointers

`const` (Zeiger)	Zeiger **Konstantheit**

Es gibt zwei Arten von Konstantheit:

read only

kein Schreibzugriff auf Target: `const int* a_ptr = &a;`

kein Schreibzugriff auf Zeiger: `int* const a_ptr = &a;`

var is const

to pointer

```
int a = 5;
int b = 8;

const int* ptr_1 = &a;
*ptr_1 = 3; // NOT valid (change target)
ptr_1 = &b; // valid (change pointer)

int* const ptr_2 = &a;
*ptr_2 = 3; // valid (change target)
ptr_2 = &b; // NOT valid (change pointer)

const int* const ptr_3 = &a;
*ptr_3 = 3; // NOT valid (change target)
ptr_3 = &b; // NOT valid (change pointer)
```

from "Summary 11" on Course Website

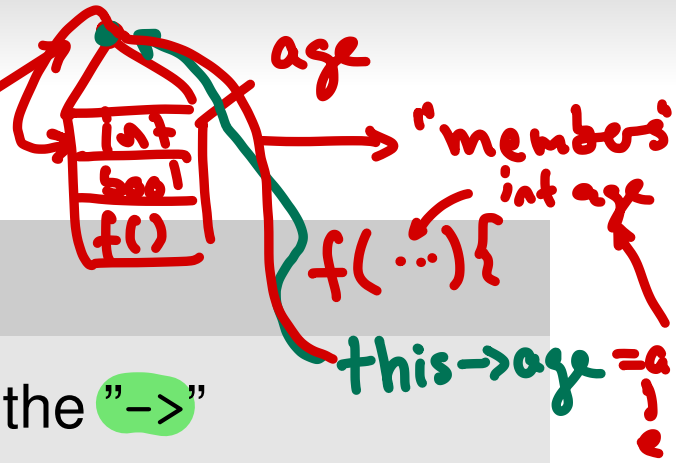
Questions?

Exercise ”Push Back”

Tasks

1. Open ”Push Back” in [code]expert
2. Try to implement it
3. On a high level this involves the following steps:
   - 3.1 Allocating a new memory block that is larger by one element.
   - 3.2 Copying all elements from the old memory block to the new one.
   - 3.3 Adding the new element to the end of the new memory block.
4. Share and discuss your implementations

What the f*&k is `this->`?



Basically[2]

- ”`this->`” has two parts: the ”`this`” and the ”`->`”
- `this` is a pointer to the current object (usually a class or struct), so it’s of type `T*`
- `->` is a very cool looking operator
  `this->member_element` is equivalent to
  `*(this).member_element` The arrow operator dereferences a pointer to an object in order to access one of its members (functions or variables)
- More details later...

[2] a word I like to preface bad explanations and oversimplifications with

Questions?

working, "easy" but inefficient Solution

```
9  void avec::push_back(int new_element) {
10
11   // allocating new memory with count + 1 many elements
12   int* new_array = new int[this->count + 1];
13
14   // copy all elements from old to new array
15   for(unsigned int it = 0; it < this->count; it++){
16     new_array[it] = this->elements[it];
17   }
18
19   // appending new element to (new) array
20   new_array[count] = new_element;
21   this->count++;
22
23   delete[] this->elements;          // optional (for now...)
24   this->elements = new_array;
25 }
```

working, "better" Solution

```
9  void avec::push_back(int new_element) {
10
11   // allocating new memory with count + 1 many elements
12   int* new_array = new int[this->count + 1];
13
14   // copy all elements from old to new array
15   const int* const src_end = this->elements + this->count + 1
16   const int* src = this->elements;
17   int* dst = new_array;
18
19   while(src != src_end){
20     *dst = *src;
21     src++;
22     dst++;
23   }
24
25   // appending new element to (new) array
26   new_array[count] = new_element;
27   this->count++;
28
29   delete[] this->elements;          // optional (for now...)
30   this->elements = new_array;
31 }
```