Exercise 7 – References, Arrays, Vectors

Informatik I für Mathematiker und Physiker (HS 2015)

Yeara Kozlov

Slides courtesy of Kaan Yücer & Endri Dibra

Agenda

- **HW #5 Feedback**
- References
  - Call by value
  - Call by reference
- Static Arrays
- Vectors
- Characters

HW #5 Feedback

- float operations using int workarounds
- indentation
- tip: handling float inputs
- good use of functions

Agenda

- HW #5 Feedback
- **References**
  - Call by value
  - Call by reference
- Static Arrays
- Vectors
- Characters

References

- Used as an alias for another variable
- Must be **initialized** with a **lvalue**
- Can only be initialized once

```
int a = 3;
int& b = a;
std::cout << b << "\n"; // Output: 3
a = 4;
std::cout << b << "\n"; // Output: 4
b = 2;
std::cout << a << "\n"; // Output: 2
```

References

- **Cannot** be initialized as an r-value

```
int& b = 3;
```

References

```
int i = 1;
int& j = i;

i++;
j++;
```

References

```
int i = 1;
int& j = i;

i++;    // i = 2
j++;
```

References

```
int i = 1;
int& j = i;

i++;    // i = 2
j++;    // i = 3
```

Call by Value

- When you pass an argument to a function, the value is copied into it:

```
void foo(int i);
int a = 5;
foo(a); //the value of a is copied to the function
```

- The value of ‚a' cannot be changed from inside the function:

```
void foo(int i) {
     i = i + 1; //Variable outside the function
                  unaffected
 }
```

Call by Value

```
void foo(int i)
{
        i = i + 1; //NO EFFECT on outside value
}

int main()
{
      int var = 5;
      foo(var);
      cout << var << endl; //output = ?
}
```

Call by Value

```
int foo(int i)
{
   return i + 1; //can return a single value
}

int main()
{
      int var = 5;
      var = foo(var);
      cout << var << endl; //output = ?
}
```

Call By Reference

- What if we need to compute and return multiple values?
- Example: computing the roots of a quadratic equations.

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Need to use call by reference

- Calling by reference - function is passed alias to the original variables.

Call By Reference

- Function definition example:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
// POST:  return value is the number of distinct real solutions of the quadratic
//        equation ax^2 + bx + c = 0. If there are infinitely many solutions
//        (a = b = c = 0), the return value is -1. Otherwise, the return value
//        is a number n from {0,1,2} and the solutions are written to s1, ... , sn
int solve_quadratic_equation (const double a, const double b, const double x,
                              double& s1, double& s2)
```

Exercise 1

Write a function `swap` that swaps the values of two `int`-variables.

Example:

```
int a = 5;
int b = 6;
// here comes your function call
std::cout << a << "\n"; // outputs 6
std::cout << b << "\n"; // outputs 5
```

Exercise 1

Solution:

```
// POST: the values of i and j are swapped
void swap (int& i, int& j) {
    const int h = i;
    i = j;
    j = h;
}
```

```
int a = 5;
int b = 6;
swap(a, b);
std::cout << a << "\n"; // outputs 6
std::cout << b << "\n"; // outputs 5
```

Exercise 1

Solution:

```
// POST: the values of i and j are swapped
void swap (int& i, int& j) {
    const int h = i;
    i = j;
    j = h;
}
```

```
int a = 5;
int b = 6;
swap(a, b);
std::cout << a << "\n"; // outputs 6
std::cout << b << "\n"; // outputs 5
```

a: 5

Exercise 1

Solution:

```
// POST: the values of i and j are swapped
void swap (int& i, int& j) {
    const int h = i;
    i = j;
    j = h;
}
```

```
int a = 5;
int b = 6;
swap(a, b);
std::cout << a << "\n"; // outputs 6
std::cout << b << "\n"; // outputs 5
```

```
a:  5
b:  6
```

Exercise 1

Solution:

```
// POST: the values of i and j are swapped
void swap (int& i, int& j) {
    const int h = i;
    i = j;
    j = h;
}
```

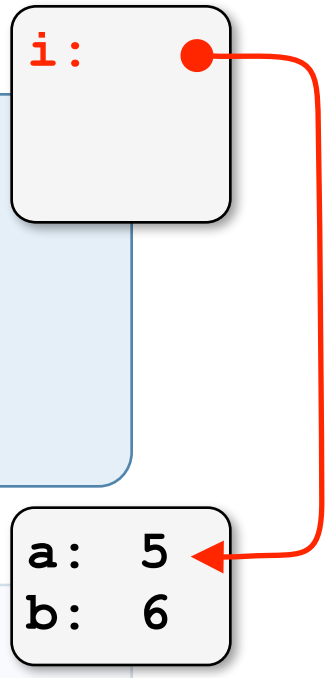
```
int a = 5;
int b = 6;
swap(a, b);
std::cout << a << "\n"; // outputs 6
std::cout << b << "\n"; // outputs 5
```

```
a:  5
b:  6
```

Exercise 1

Solution:

```
// POST: the values of i and j are swapped
void swap (int& i, int& j) {
    const int h = i;
    i = j;
    j = h;
}
```

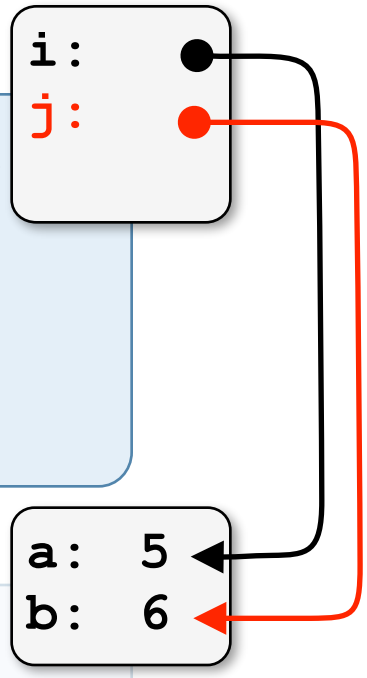



```
int a = 5;
int b = 6;
swap(a, b);
std::cout << a << "\n"; // outputs 6
std::cout << b << "\n"; // outputs 5
```

Exercise 1

Solution:

```
// POST: the values of i and j are swapped
void swap (int& i, int& j) {
    const int h = i;
    i = j;
    j = h;
}
```

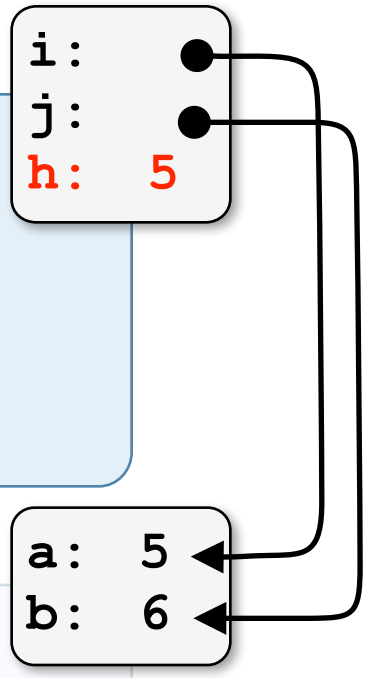



```
int a = 5;
int b = 6;
swap(a, b);
std::cout << a << "\n"; // outputs 6
std::cout << b << "\n"; // outputs 5
```

Exercise 1

Solution:

```
// POST: the values of i and j are swapped
void swap (int& i, int& j) {
    const int h = i;
    i = j;
    j = h;
}
```

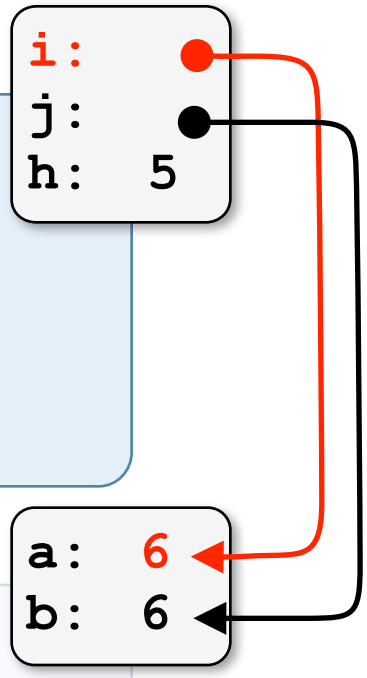



```
int a = 5;
int b = 6;
swap(a, b);
std::cout << a << "\n"; // outputs 6
std::cout << b << "\n"; // outputs 5
```

Exercise 1

Solution:

```
// POST: the values of i and j are swapped
void swap (int& i, int& j) {
    const int h = i;
    i = j;
    j = h;
}
```

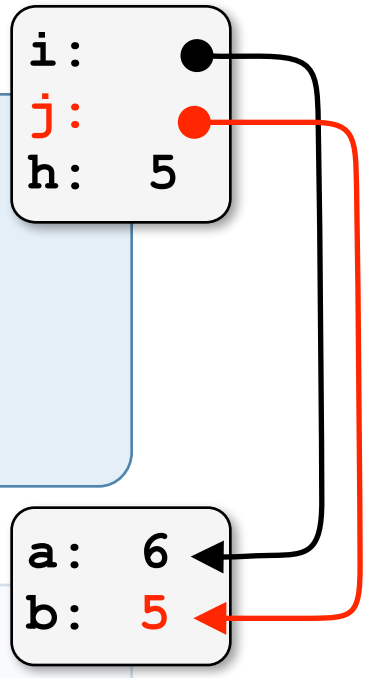



```
int a = 5;
int b = 6;
swap(a, b);
std::cout << a << "\n"; // outputs 6
std::cout << b << "\n"; // outputs 5
```

Exercise 1

Solution:

```
// POST: the values of i and j are swapped
void swap (int& i, int& j) {
    const int h = i;
    i = j;
    j = h;
}
```




```
int a = 5;
int b = 6;
swap(a, b);
std::cout << a << "\n"; // outputs 6
std::cout << b << "\n"; // outputs 5
```

Exercise 1

Solution:

```
// POST: the values of i and j are swapped
void swap (int& i, int& j) {
    const int h = i;
    i = j;
    j = h;
}
```

```
int a = 5;
int b = 6;
swap(a, b);
std::cout << a << "\n"; // outputs 6
std::cout << b << "\n"; // outputs 5
```

```
a:  6
b:  5
```

Exercise 1

Solution:

```
// POST: the values of i and j are swapped
void swap (int& i, int& j) {
    const int h = i;
    i = j;
    j = h;
}
```

```
int a = 5;
int b = 6;
swap(a, b);
std::cout << a << "\n"; // outputs 6
std::cout << b << "\n"; // outputs 5
```

```
a:  6
b:  5
```

Exercise 2

(a)

What is the output of the program for the following variant of `foo`?

```
int foo (int& a, int b) {
  a += b;
  return a;
}
```

```
int main() {
  int a = 0;
  int b = 1;
  for (int i=0; i<5; ++i) {
    b = foo (a, b);
    std::cout << b << " ";
  }
  return 0;
}
```


(From: Exam Summer 2015, Exercise 9)

Exercise 2

(a)

What is the output of the program for the following variant of `foo`?

```
1 2 4 8 16
```

```
int foo (int& a, int b) {
  a += b;
  return a;
}
```

```
int main() {
  int a = 0;
  int b = 1;
  for (int i=0; i<5; ++i) {
    b = foo (a, b);
    std::cout << b << " ";
  }
  return 0;
}
```


(From: Exam Summer 2015, Exercise 9)

Exercise 2

(b)

What is the output of the program for the following variant of `foo`?

```
int foo (int a, int b) {
  a += b;
  return a;
}
```

```
int main() {
  int a = 0;
  int b = 1;
  for (int i=0; i<5; ++i) {
    b = foo (a, b);
    std::cout << b << " ";
  }
  return 0;
}
```


(From: Exam Summer 2015, Exercise 9)

Exercise 2

(b)

What is the output of the program for the following variant of `foo`?

```
1 1 1 1 1
```

```
int foo (int a, int b) {
  a += b;
  return a;
}
```

```
int main() {
  int a = 0;
  int b = 1;
  for (int i=0; i<5; ++i) {
    b = foo (a, b);
    std::cout << b << " ";
  }
  return 0;
}
```


(From: Exam Summer 2015, Exercise 9)

Exercise 2

(c)

What is the output of the program for the following variant of `foo`?

```
int foo (int a, int& b) {
  a += b;
  return a;
}
```

```
int main() {
  int a = 0;
  int b = 1;
  for (int i=0; i<5; ++i) {
    b = foo (a, b);
    std::cout << b << " ";
  }
  return 0;
}
```


(From: Exam Summer 2015, Exercise 9)

Exercise 2

(c)

What is the output of the program for the following variant of `foo`?

```
1 1 1 1 1
```

```
int foo (int a, int& b) {
  a += b;
  return a;
}
```

```
int main() {
  int a = 0;
  int b = 1;
  for (int i=0; i<5; ++i) {
    b = foo (a, b);
    std::cout << b << " ";
  }
  return 0;
}
```


(From: Exam Summer 2015, Exercise 9)

Return by Reference

```
int& increment (int& m) {
  return ++m;
}

int main () {
  int n = 3;
  increment (increment (n));
  return 0;
}
```

- Return value here is int &
- Final value of n: 5
- Operator ++ is structured the same way: ++(++i)

Agenda

- HW #5 Feedback
- References
  - Call by value
  - Call by reference
- **Static Arrays**
- Vectors
- Characters

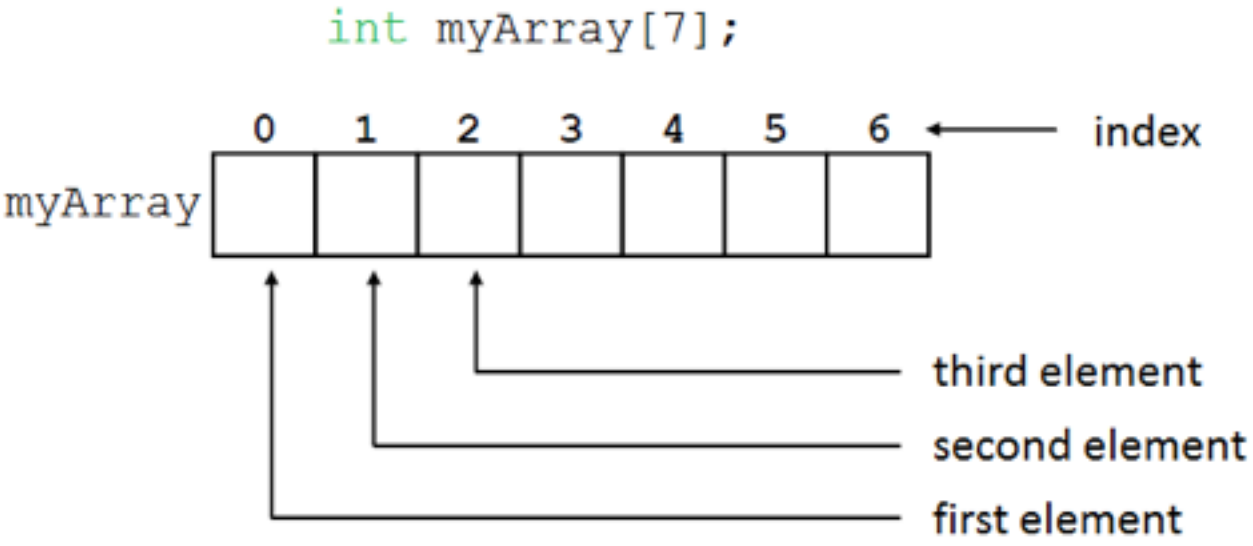
Static Arrays

- Primitive C data-type
- Store multiple values of the same type, e.g. `int`

- Definition consists of three elements:
  - Data type
  - Name
  - Number of elements - must be known at compile time

```
typeName arrayName[arraySize]
int myArray[100];
```

Static Arrays



Static Arrays

- Checking the array bounds in the programmer's responsibility.

```
int a[] = {7,5,0,3,8};
std::cout << a[0];       // outputs 7
std::cout << a[4];       // outputs 8
std::cout << a[5];       // outputs random garbage (or sometimes segmentation fault)
std::cout << a[-10];     // outputs random garbage (or sometimes segmentation fault)
```

Arrays - Initialisation and Assignment

- Initialisation at definition:

```
int a[5] = {0,1,2,3,4};
```

  - Explicitly

```
int b[3];
b[0] = 7;
```

- Partial Initialisation

```
int a[5] = {1,2};
```

  - Other elements are automatically set to 0

- Let the compiler sets the size:

```
int a[] = {1,2,3};
```

Reading / Writing into Arrays – loops

```
int numbers[10];
```

Reading / Writing into Arrays – loops

```
int numbers[10];

for (int i = 0; i < 10; i++)
  std::cin >> numbers[i];
```

Reading / Writing into Arrays – loops

```
int numbers[10];

for (int i = 0; i < 10; i++)
  std::cin >> numbers[i];


for (int i = 0; i < 10; i++)
  std::cout << numbers[i] << " ";
std::cout << "\n";
```

Array to array assignment

- This does not work

```
int a[] = {7,5,0,3,8};
int b[5];
b = a;    // does not compile
```

Array to array assignment

- This does not work

```
int a[] = {7,5,0,3,8};
int b[5];
b = a;    // does not compile
```

- Use loops instead

Array to array assignment

- This does not work

```
int a[] = {7,5,0,3,8};
int b[5];
b = a;    // does not compile
```

- Use loops instead

```
int a[] = {7,5,0,3,8};
int b[5];

for (int i = 0; i < 5; i++)
  b[i] = a[i];
```

Exercise : Reverse Array

- Write a program which performs the following steps:
    1. Define an array of type `int` and of length 20.
    2. Fill it with numbers from `std::cin`.
    3. Reverse the array.
    4. Write reversed array to output

```cpp
#include <iostream>

int main () {

    return 0;
}
```

Write a program which performs the following steps:

1. Define an array of type `int` and of length 20.
2. Fill it with numbers from `std::cin`.
3. Reverse the array.
4. Output this.

```cpp
#include <iostream>

int main () {
    const int len = 20;

    // Step 1: Define array
    int input[len];

    return 0;
}
```

Write a program which performs the following steps:

1. Define an array of type `int` and of length 20.
2. Fill it with numbers from `std::cin`.
3. Reverse the array.
4. Output this.

Write a program which performs the following steps:

1. Define an array of type `int` and of length 20.
2. Fill it with numbers from `std::cin`.
3. Reverse the array.
4. Output this.

```cpp
#include <iostream>

int main () {
    const int len = 20;

    // Step 1: Define array
    int input[len];

    // Step 2: Read from std::cin
    for (int i = 0; i < len; ++i)
        std::cin >> input[i];

    return 0;
}
```

```cpp
#include <iostream>

// POST: the targets of i and j got their values swapped
void swap (int& i, int& j) {
    const int tmp = i;
    i = j;
    j = tmp;
}

int main () {
    const int len = 20;

    // Step 1: Define array
    int input[len];

    // Step 2: Read from std::cin
    for (int i = 0; i < len; ++i)
        std::cin >> input[i];









    return 0;
}
```

Write a program which performs the following steps:

1. Define an array of type `int` and of length 20.
2. Fill it with numbers from `std::cin`.
3. Reverse the array.
4. Output this.

```cpp
#include <iostream>

// POST: the targets of i and j got their values swapped
void swap (int& i, int& j) {
    const int tmp = i;
    i = j;
    j = tmp;
}

int main () {
    const int len = 20;

    // Step 1: Define array
    int input[len];

    // Step 2: Read from std::cin
    for (int i = 0; i < len; ++i)
        std::cin >> input[i];

    // Step 3: Reverse array
    int front = 0;
    int back = len-1;
    while (front < back) {
        swap(input[front], input[back]);
        ++front;
        --back;
    }




    return 0;
}
```

Write a program which performs the following steps:

1. Define an array of type `int` and of length 20.
2. Fill it with numbers from `std::cin`.
3. Reverse the array.
4. Output this.

```cpp
#include <iostream>

// POST: the targets of i and j got their values swapped
void swap (int& i, int& j) {
    const int tmp = i;
    i = j;
    j = tmp;
}

int main () {
    const int len = 20;

    // Step 1: Define array
    int input[len];

    // Step 2: Read from std::cin
    for (int i = 0; i < len; ++i)
        std::cin >> input[i];

    // Step 3: Reverse array
    int front = 0;
    int back = len-1;
    while (front < back) {
        swap(input[front], input[back]);
        ++front;
        --back;
    }
    // Step 4: Output array
    for (int i = 0; i < len; ++i)
        std::cout << input[i] << "  ";
    std::cout << "\n";

    return 0;
}
```

Write a program which performs the following steps:

1. Define an array of type `int` and of length 20.
2. Fill it with numbers from `std::cin`.
3. Reverse the array.
4. Output this.

Agenda

- HW #5 Feedback
- References
  - Call by value
  - Call by reference
- Static Arrays
- **Vectors**
- Characters

Vectors

- Determine length at runtime
- Use `std::vector`
- `#include <vector>`

- Initialize

```
std::vector<int> my_vec (my_length, my_start_value);
```

- Instead of

```
int my_arr[my_length];
```

Vectors

Improve your program from the array exercise so that an arbitrarily long sequence can be reversed.

The length of the sequence is the first input.

```cpp
#include <iostream>

// POST: the targets of i and j got their values swapped
void swap (int& i, int& j) {
    const int tmp = i;
    i = j;
    j = tmp;
}

int main () {
    const int len = 20;

    // Step 1: Define array
    int input[len];

    // Step 2: Read from std::cin
    for (int i = 0; i < len; ++i)
        std::cin >> input[i];

    // Step 3: Reverse array
    int front = 0;
    int back = len-1;
    while (front < back) {
        swap(input[front], input[back]);
        ++front;
        --back;
    }
    // Step 4: Output array
    for (int i = 0; i < len; ++i)
        std::cout << input[i] << "  ";
    std::cout << "\n";

    return 0;
}
```

- Improve your program from the array exercise so that an arbitrarily long sequence can be reversed.

- The length of the sequence shall be given as the first input to your program.

```cpp
#include <iostream>
#include <vector>                          //$ (this is also new)
// POST: the targets of i and j got their values swapped
void swap (int& i, int& j) {
    const int tmp = i;
    i = j;
    j = tmp;
}

int main () {
    const int len = 20;

    // Step 1: Define array
    int input[len];

    // Step 2: Read from std::cin
    for (int i = 0; i < len; ++i)
        std::cin >> input[i];

    // Step 3: Reverse array
    int front = 0;
    int back = len-1;
    while (front < back) {
        swap(input[front], input[back]);
        ++front;
        --back;
    }
    // Step 4: Output array
    for (int i = 0; i < len; ++i)
        std::cout << input[i] << "  ";
    std::cout << "\n";

    return 0;
}
```

- Improve your program from the array exercise so that an arbitrarily long sequence can be reversed.

- The length of the sequence shall be given as the first input to your program.

```cpp
#include <iostream>
#include <vector>                          //$ (this is also new)
// POST: the targets of i and j got their values swapped
void swap (int& i, int& j) {
    const int tmp = i;
    i = j;
    j = tmp;
}

int main () {
    int len;                               //$ const int len = 20;
    std::cin >> len;                       //$ (this is also new)
    // Step 1: Define array
    std::vector<int> input (len);          //$ int input[len];

    // Step 2: Read from std::cin
    for (int i = 0; i < len; ++i)
        std::cin >> input[i];

    // Step 3: Reverse array
    int front = 0;
    int back = len-1;
    while (front < back) {
        swap(input[front], input[back]);
        ++front;
        --back;
    }
    // Step 4: Output array
    for (int i = 0; i < len; ++i)
        std::cout << input[i] << "  ";
    std::cout << "\n";

    return 0;
}
```

- Improve your program from the array exercise so that an arbitrarily long sequence can be reversed.

- The length of the sequence shall be given as the first input to your program.

Agenda

- HW #5 Feedback
- References
  - Call by value
  - Call by reference
- Static Arrays
- Vectors
- **Characters**

Characters - ASCII

`char` is a primitive C type which stores a single character.

```
char ch = 'z'; // !single! quotes

// read 5 chars from user and count number of 'n's

unsigned int counter = 0;
for (int i=0; i<5; ++i)
{
    std::cin >> ch;
    if (ch == 'n')
        ++counter;
}
std::cout << counter << "\n";
```

HW #6 Pre-discussion

- Exercise 1 - explanation needed!
- Exercise 2 - do you need to store these movements?