ETH zürich

Exercise Session W07

Computer Science (CSE & CBBI & Statistics) – AS 23

Overview

Today's Agenda

Follow-up
Objectives
References
`std::vector<T>`
(ASCII) Characters
Feedback
Repetition: Floating Point Numbers
Outro

`n.ethz.ch/~agavranovic`

1. Follow-up

Follow-up

- I added some slides to last week's folder that I forgot to upload last time (can be found under *Addendum*))
- You can see my changes when you click "View Submission"
  - If you still can't see them: then email me

2. Objectives

Objectives

- ☐ be able to trace and write programs that use references
- ☐ be able to write programs that create, modify, and iterate over vectors
- ☐ be able to trace and write programs that modify ASCII characters

3. References

Example of Program Tracing I

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

b = 2;

std::cout << a;
```

Output:

Example of Program Tracing I

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

b = 2;

std::cout << a;
```

Output: 2

Example of Program Tracing II

```
void foo(int i){
    i = 5;
}

int main(){
    int i = 4;
    foo(i);
    std::cout << i << std::endl;
}
```

Output:

Example of Program Tracing II

```
void foo(int i){
    i = 5;
}

int main(){
    int i = 4;
    foo(i);
    std::cout << i << std::endl;
}
```

Output: 4 ... but why?

Example of Program Tracing II

```
void foo(int i){
    i = 5;
}

int main(){
    int i = 4;
    foo(i);
    std::cout << i << std::endl;
}
```

Output: `4` …but why? References (`type&`) are used as type of function parameters (inputs) or return types (returns). If the parameters are **not** *referenced*, we say *passed to the function by value*. (This is how we did it for all previous functions). This always makes a copy of the input to the function.

Example of Program Tracing III

```
void foo(int& i){
    i = 5;
}

int main(){
    int i = 4;
    foo(i);
    std::cout << i << std::endl;
}
```

Output:

Example of Program Tracing III

```
void foo(int& i){
    i = 5;
}

int main(){
    int i = 4;
    foo(i);
    std::cout << i << std::endl;
}
```

Output: 5

Example of Program Tracing III

```
void foo(int& i){
    i = 5;
}

int main(){
    int i = 4;
    foo(i);
    std::cout << i << std::endl;
}
```

Output: 5 When a function parameter is a reference type (`type&`), we say "*passed (the argument) by reference*"

References

Why all this?

References

Why all this?

- you can influence several results/variables and don't have to rely on the `return`

References

Why all this?

- you can influence several results/variables and don't have to rely on the `return`
- you can save the (sometimes expensive) copying of parameters and thus improve the performance of the program.

References

Why all this?

- you can influence several results/variables and don't have to rely on the `return`
- you can save the (sometimes expensive) copying of parameters and thus improve the performance of the program.
- sometimes there is no other way (`std::cout` for example, we will have a look in a few weeks)

Questions?

References as Return Types

We have now seen function parameters that have a reference type, but references can also be used for return types

References as Return Types

We have now seen function parameters that have a reference type, but references can also be used for return types

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

int main(){
    int n = 3;

    increment(increment(n));

    std::cout << n << std::endl;
}
```

Output:

References as Return Types

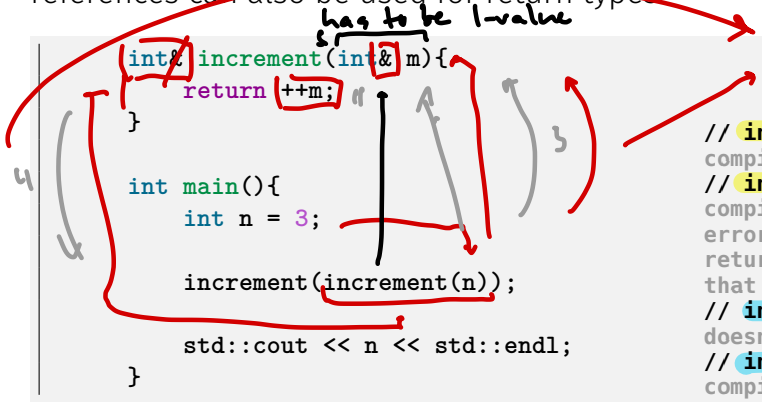
We have now seen function parameters that have a reference type, but references can also be used for return types

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

int main(){
    int n = 3;

    increment(increment(n));

    std::cout << n << std::endl;
}
```

Output: 5, but why?

References as Return Types

We have now seen function parameters that have a reference type, but references can also be used for return types.




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

int main(){
    int n = 3;

    increment(increment(n));

    std::cout << n << std::endl;
}
```

```
// int& increment(int& m)
compiles, output 5
// int& increment(int m)
compiles, but gives segmentation
error (because we're trying to
return a reference to something
that "died")
// int increment(int& m)
doesn't even compile
// int increment(int m){
compiles, output 3
```

Output: 5, but why? Because of the references!

Questions?

Reference or Copy? I

```
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;
}
```

Output:

Reference or Copy? I

```
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;
}
```

Output: 1 2 4 8 16

Reference or Copy? I

```
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;
}
```

Output: 1 2 4 8 16... buy why?

Reference or Copy? II

```
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;
}
```

Output:

Reference or Copy? II

```
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;
}
```

Output: `1 1 1 1 1`

Reference or Copy? II

```
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;
}
```

Output: 1 1 1 1 1… buy why?

Reference or Copy? III

error in slide. if there were a b it would have output...

```
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;
}
```

Output:

Reference or Copy? III

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;
}
```

Output: 1 1 1 1 1

Reference or Copy? III

```
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;
}
```

Output: 1 1 1 1 1… buy why?

Questions?

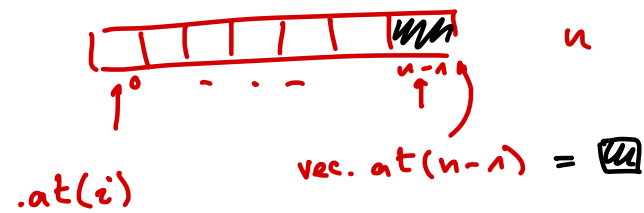
4. std::vector<T>

How to `std::vector`

- `#include <vector>`

How to `std::vector`

- `#include <vector>`
- Vectors can be thought of as a series of boxes, each storing a value of the given type



How to `std::vector`

- `#include <vector>`
- Vectors can be thought of as a series of boxes, each storing a value of the given type
- You can treat vectors something like a new type

How to `std::vector`

- `#include <vector>`
- Vectors can be thought of as a series of boxes, each storing a value of the given type
- You can treat vectors something like a new type
- `std::vector<int> myvector{1,2,3};`
  to initialize a vector

How to `std::vector`

- `#include <vector>`
- Vectors can be thought of as a series of boxes, each storing a value of the given type
- You can treat vectors something like a new type
- `std::vector<int> myvector{1,2,3};`
  to initialize a vector
- There are many ways to initialize/define a vector. Look in the Summaries or search online

How to `std::vector`

- `#include <vector>`
- Vectors can be thought of as a series of boxes, each storing a value of the given type
- You can treat vectors something like a new type
- `std::vector<int> myvector{1,2,3};`
  to initialize a vector
- There are many ways to initialize/define a vector. Look in the Summaries or search online
- `myvector[n-1]` ~~`n-1`~~ `.at(n-1)` `.at(i)`
  to get the n'th value in the vector

How to `std::vector`

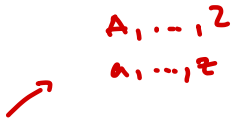
- `#include <vector>`
- Vectors can be thought of as a series of boxes, each storing a value of the given type
- You can treat vectors something like a new type
- `std::vector<int> myvector{1,2,3};`
  to initialize a vector
- There are many ways to initialize/define a vector. Look in the Summaries or search online
- `myvector[n-1]`. `at(n-1)`
  to get the n'th value in the vector
- `myvector.push_back(x)`
  to append the value `x`

variable called n

`if(n == 'n'){...}`

the char n

Questions?



5. (ASCII) Characters

Exercise "Converting Input to UPPER CASE"

Task

uns int diff = 'a' - 'A'

Write a program that reads a sequence of characters, delimited by the new-line character, as a vector of `char`. Then the program should output the sequence with all lower-case letters changed to UPPER-CASE letters.
To read the sequence you can:

- read a single character from standard input
- insert it into a vector of chars
- repeat until you find a newline character (`\n`).

Please put the code that converts the entire sequence to upper-case and a single character to upper-case into separate functions (you should have at least three functions).
Hint: variables of type char can be treated as numbers.

ABC...2 #!?..@abc...z

Exercise "Converting Input to UPPER CASE"

Task

1. Consider how best to approach the "Converting Input to UPPER CASE" task on **code** expert

Exercise "Converting Input to UPPER CASE"

Task

1. Consider how best to approach the "Converting Input to UPPER CASE" task on **code** expert
2. Implement (optionally in groups) a solution

(Solution) "Converting Input to UPPER CASE"

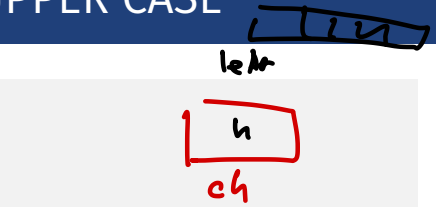
```
#include <iostream>
#include <vector>
#include <ios>
```

(Solution) "Converting Input to UPPER CASE"

```
// POST: Converts the letter to upper case.
void char_to_upper(char& letter){
    if('a' <= letter && letter <= 'z'){
        letter -= 'a' - 'A';  // 'a' > 'A'
    }
}

// POST: Converts all letters to upper-case.
void to_upper(std::vector<char>& letters){
    for(unsigned int i = 0; i < letters.size(); ++i){
        char_to_upper(letters.at(i));
    }
}
```

Solution "Converting Input to UPPER CASE"

```
std::cin >> std::noskipws;
std::vector<char> letters;
char ch;

// Step 1: Read input.
do{
    std::cin >> ch;
    letters.push_back(ch);
}while(ch != '\n');

// Step 2: Convert to upper-case.
to_upper(letters);

// Step 3: Output.
for(unsigned int i = 0; i < letters.size(); ++i){
    std::cout << letters.at(i);
}
```



Questions?

6. Feedback

Your Feedback to me

Feedback form

(Take your time and be frank)

7. Repetition: Floating Point Numbers

Normalized Floating Point Number Systems

Task

- Try to solve following tasks (as a group)
- Ask if anything remain unclear

Consider the normalized floating point number system $F^*(\beta, p, e_{\min}, e_{\max})$ with $\beta = 2$, $p = 3$, $e_{\min} = -4$, $e_{\max} = 4$.
Compute the following expressions as the parentheses suggest, representing each intermediate result (and the final result) in the normalized floating point system according to the rules of computing with floating point numbers.

$(10 + 0.5) + 0.5$

	decimal		binary
	10		?????
+	0.5		?????
=			?????
+	0.5		?????
=	??	←	?????

$(0.5 + 0.5) + 10$

	decimal		binary
	0.5		?????
+	0.5		?????
=			?????
+	10		?????
=	??	←	?????

$(10 + 0.5) + 0.5$

	decimal		binary
	10		$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$			?????
$+$	0.5		?????
$=$	??	$\leftarrow$	?????

$(0.5 + 0.5) + 10$

	decimal		binary
	0.5		?????
$+$	0.5		?????
$=$			?????
$+$	10		?????
$=$	??	$\leftarrow$	?????

$(10 + 0.5) + 0.5$

	decimal		binary
	10		$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$			$1.0101 \cdot 2^3$
$+$	0.5		?????
$=$	??	$\leftarrow$	?????

$(0.5 + 0.5) + 10$

	decimal		binary
	0.5		?????
$+$	0.5		?????
$=$			?????
$+$	10		?????
$=$	??	$\leftarrow$	?????

$(10 + 0.5) + 0.5$

	decimal		binary
	10		$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$			$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$	??	$\leftarrow$	?????

$(0.5 + 0.5) + 10$

	decimal		binary
	0.5		?????
$+$	0.5		?????
$=$			?????
$+$	10		?????
$=$	??	$\leftarrow$	?????

$(10 + 0.5) + 0.5$

	decimal		binary
	10		$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$			$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$	10	$\leftarrow$	$1.01 \cdot 2^3$

$(0.5 + 0.5) + 10$

	decimal		binary
	0.5		?????
$+$	0.5		?????
$=$			?????
$+$	10		?????
$=$	??	$\leftarrow$	?????

$(10 + 0.5) + 0.5$

	decimal		binary
	10		$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$			$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$	10	$\leftarrow$	$1.01 \cdot 2^3$

$(0.5 + 0.5) + 10$

	decimal		binary
	0.5		$1.00 \cdot 2^{-1}$
$+$	0.5		$1.00 \cdot 2^{-1}$
$=$			?????
$+$	10		?????
$=$	??	$\leftarrow$	?????

$(10 + 0.5) + 0.5$

	decimal		binary
	10		$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$			$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$	10	$\leftarrow$	$1.01 \cdot 2^3$

$(0.5 + 0.5) + 10$

	decimal		binary
	0.5		$1.00 \cdot 2^{-1}$
$+$	0.5		$1.00 \cdot 2^{-1}$
$=$			$1.00 \cdot 2^0$
$+$	10		$1010.00 \cdot 2^0$
$=$	??	$\leftarrow$	?????

$(10 + 0.5) + 0.5$

	decimal		binary
	10		$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$			$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$	10	$\leftarrow$	$1.01 \cdot 2^3$

$(0.5 + 0.5) + 10$

	decimal		binary
	0.5		$1.00 \cdot 2^{-1}$
$+$	0.5		$1.00 \cdot 2^{-1}$
$=$			$1.00 \cdot 2^0$
$+$	10		$1010.00 \cdot 2^0$
$=$	??	$\leftarrow$	$1011.00 \cdot 2^0$

$(10 + 0.5) + 0.5$

	decimal		binary
	10		$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$			$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$	10	$\leftarrow$	$1.01 \cdot 2^3$

$(0.5 + 0.5) + 10$

	decimal		binary
	0.5		$1.00 \cdot 2^{-1}$
$+$	0.5		$1.00 \cdot 2^{-1}$
$=$			$1.00 \cdot 2^0$
$+$	10		$1010.00 \cdot 2^0$
$=$	??	$\leftarrow$	$1.011 \cdot 2^3$

1.100

$\downarrow$

$1.10 \cdot 2^3$

$(10 + 0.5) + 0.5$

	decimal		binary
	10		$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$			$1.01 \cdot 2^3$
$+$	0.5		$0.0001 \cdot 2^3$
$=$	10	$\leftarrow$	$1.01 \cdot 2^3$

$(0.5 + 0.5) + 10$

	decimal		binary
	0.5		$1.00 \cdot 2^{-1}$
$+$	0.5		$1.00 \cdot 2^{-1}$
$=$			$1.00 \cdot 2^0$
$+$	10		$1010.00 \cdot 2^0$
$=$	12	$\leftarrow$	$1.10 \cdot 2^3$

Questions?

8. Outro

General Questions?

re: "constness" codeexpert exercise:

→ if no `const` is used in the code snippet, then it's "trivially" respected

Till next time!

Cheers!