ETH zürich

Session 06
Reference Types, Vectors

Informatik I

Lan Zhang

27.03.2025

Schedule

1. Review of Last Week
2. Main Topics
   - References
   - Vectors
3. In-Class Code Example
4. Practice Questions

References

An alias for an already existing object: `Type& alias = x; \\ alias is new name for x`

A reference stores the address of i.e. occupies the same memory location as the object it refers to.

```
int a = 3;
int& b = a; // b has value &a
b = 7;
std::cout << a << std::endl; // outputs 7
```

Const-References: can be initialized with L- or R-values (since not modifiable)

```
const float pi = 3.1415926f; // read only reference
float& a = pi;       // invalid: const-qualification discarded
const float& a = pi; // valid
(const T var = T const var)
```

References

Used as Function Parameters or Return Values

- Called “pass by reference”
- Changes made to variable within function will persist

Pass by value

```
void foo (int i) {
...
}
```

Function uses its own copies of the passed parameters, which are deleted when scope ends

Pass by reference

```
void foo (int& i) {
...
}
```

Function directly operates on the call arguments

Return by reference

```
int& foo (int& i) {
...
}
```

Function returns a reference to the return value

```
int& preincrement
(int& x)

return ++x;
```

References

The object to which a reference refers must “exist” at least as long as the reference itself

```
int& foo(int i) {
  return i;
}


int k = 3;
int& j = foo(k);
std::cout << j;


In function 'int& foo(int)':
warning: reference to local variable 'i' returned [-Wreturn-local-addr]
```

References: Examples

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; }
```

a passed as reference

i:	0 1 2 3 4
a:	1 2 4 8 16
b:	1 2 4 8 16

References: Examples

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; }
```

a passed by value

i:	0 1 2 3 4
a:	-- 0 --
b:	1 1 1 1 1

References: Examples

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; }
```

b passed by reference, but not changed in foo()

i: 0 1 2 3 4

a: -- 0 --

b: 1 1 1 1 1

Vectors

- Data structure that holds many instances of some data type
- Occupies contiguous memory area
- Declaration:

```
#include <vector>
std::vector<T> vec; // empty vector

std::vector<T> v(n);// vector of size n with value 0
std::vector<T> v(n, x);// vector of size n with value x
std::vector<T> v{a,b,c,d}; // vector initialized as list of elements
```

- access to elements (Indexing starts from 0) -> l-values

```
std::cout << v[n]; // undefined behavior (random value)
std::cout << v.at(n); // throws exception std::out_of_range
```

Vector Interface

`v.size()` – gets number of elements in v

`v.at(i)` – returns element at index i (same as v[i], but checks whether i is out of bounds)

`v.insert(pos, val)` – inserts val at index pos

`v.push_back(val)` – inserts val at the end

`v.empty()` – returns boolean indicating whether vector contains elements

Multidimentional Vectors

```
std::vector<std::vector<T>> matrix(nrows,std::vector(ncols, init_value);)
matrix.at(row_index) // Accessing vector<T> (entire row)
matrix.at(row_index).at(col_index) // Accessing T (single element)
```

Type alias (Abbreviation of a type name)

```
using matrix = std::vector<std::vector<int>>;
matrix m = ...; // or use auto
```

In-Class Code Example

Reversing Vectors & Matrix Transpose

Pass by reference is more resource efficient as pass by value, as no local copies have to be made.

Can you think of other use-cases for references? What can we not achieve with non-reference types?

- return more than one result from a function

```
int solve_quadratic_equation(const double a, const double b, const double c,
double& x1, double& x2)
```

- for function arguments that cannot be copied

```
std::ostream o = std::cout; // Error: copying std::cout is impossible
std::ostream& o = std::cout; // This works
```

Practice Questions

What is the output of the following program?

```
#include<iostream>
#include<vector>
void increment(int& a){
     ++a;
}
int main(){
     std::vector vec = std::vector(1,0);

     for(int i = 0; i < 6; ++i){
            if(i%2 == 0) vec.push_back(i);
     }
     for(int i = 0; i < vec.size(); ++i) std::cout << increment(vec.at(i));
    }
```

Sol.: 1135

Practice Questions

What is the output of the following program?

```
#include<iostream>
#include<vector>
void foo(std::vector& a, int b){
     if(!(b < a.size())) return;
     foo(a,b+1);
     std::cout << a.at(b);
}
int main(){
     std::vector vec = {3, 1, 9, 7};
     foo(vec,0);
     return 0;
}
```

Sol.: 7913

Practice Questions

What is the output of the following program?

```
#include<iostream>
void foo(int& n, int& m){
     if (n == 0) return;
     m += n % 10;
     n /= 10;
     foo(n, m);
}
int main(){
     int m=0; int n = 5468;
     foo(n,m);
     std::cout << m << std::endl;
     return 0;
}
```

Sol.: 23