C++ References

Gordon Erlebacher ++ Peter Beerli

Area in memory

- A reference "points" to an area in memory
- A reference is **not** a memory address
- A reference **cannot** exist in isolation <u>except</u> when
  - reference is an attribute of a class
  - in which case, it is initialized in a <u>*special way*</u>
- A reference is a glorified pointer

References must be initialized!

(except in special cases involving classes)

```
#include <stdio.h>

int main()
{
    float &a;
    return 0;
}
```

Compile-time error

```
nagal:L17>clang++ references_test.cpp -o ref
references_test.cpp:13:12: error: declaration of reference
variable 'a' requires
      an initializer
    float &a;
           ^
1 error generated.
```

References must be initialized!

(except in special cases involving classes)

```
#include <stdio.h>

int main()
{
    float &a;
    return 0;
}
```

Compile-time error

```
nagal:L17>clang++ references_test.cpp -o ref
references_test.cpp:13:12: error: declaration of reference
variable 'a' requires
      an initializer
    float &a;
           ^
1 error generated.
```

```
#include <stdio.h>

int main()
{
    float b = 3.0;
    float &a = b;
    return 0;
}
```

Compiles fine!

BUT

The number 3 is an immutable object. Therefore, the reference must be a const

Const is used to indicate that something is immutable

```
int test()
{
    // Reference to the integer 3
    const int& a = 3;  // OK
    int& b = 3;        // not OK

    int c = 4;
    int& cr = b;       // OK

    return 0;
}
```

```
nagal:L17>clang++ references_test.cpp -o ref
references_test.cpp:17:10: error: non-const lvalue reference to type
'int'
      cannot bind to a temporary of type 'int'
    int& b = 3;         // not OK
         ^   ~
1 error generated.
```

No undoing

- Once a reference is linked to an area in memory, it cannot be unlinked or changed!

- This is probably the most important lesson on references!

```cpp
int test2()
{
    int vara = 3;
    int& var_r = vara;

    int varb = 5.;
    var_r = varb;

    // What is does var_r reference?
    // Answer: var_r refers to a!!!
    printf("vara= %d\n", vara); // 5
    printf("var_r= %d\n", var_r);

    varb = 7;
    printf("vara= %d\n", vara); // 5
    printf("var_r= %d\n", var_r);

    vara = 6;
    printf("vara= %d\n", vara); // 5
    printf("var_r= %d\n", var_r);

    return 0;
}
```

Compile and Run

```
nagal:L17>clang++ references_test.cpp -o ref
```

```int test2()
{
    int vara = 3;
    int& var_r = vara;

    int varb = 5.;
    var_r = varb;

    // What is does var_r reference?
    // Answer: var_r refers to a!!!
    printf("vara= %d\n", vara); // 5
    printf("var_r= %d\n", var_r);

    varb = 7;
    printf("vara= %d\n", vara); // 5
    printf("var_r= %d\n", var_r);

    vara = 6;
    printf("vara= %d\n", vara); // 6
    printf("var_r= %d\n", var_r);

    return 0;
}```

Output

```nagal:L17>ref
vara= 5
var_r= 5
vara= 5
var_r= 5
vara= 6
var_r= 6```

Compile

```nagal:L17>clang++ references_test.cpp -o ref```
```

Why use references?

- Avoid use of * and & operators when calling functions

Two ways to return a changed value: a reference and a pointer

```
void change(int ar, int* ap)
{
    ar = 134;
    *ap = 234;
}

int test3()
{
    int a = 6;
    int& ar = a;
    int* ap = &a;

    printf("before: ar,ap= %d, %d\n", ar, *ap);
    change(ar, ap);
    printf("after: ar,ap= %d, %d\n", ar, *ap);

    return 0;
}
```

```
nagal:L17>clang++ references_test.cpp -o ref
nagal:L17>ref
before: ar,ap= 6, 6
after: ar,ap= 234, 234
```

References and arrays

```
int test4()
{
    float* a = new float [10];
    float& ar = *a;

    a[5] = 7.;
    a[0] = 8.;

    printf(" a[5] = %f\n",     a[5]);
    printf("ar[5] = %f\n", (&ar)[5]);
    printf("ar = %f\n", ar);
    return 0;
}
```

```
./ref
 a[5] = 7.000000
ar[5] = 7.000000
ar = 8.000000
```

References do not help much in this case

References and Classes

- We first define a simple class
- We access some of its methods with pointers and with references
- We call a function with a class argument
- We note the advantage of references

Simple class

```
class Array2D
{
public:
    int n, m;
    float* data;

    Array2D(int n, int m) //constructor
    {
        this->n = n;
        this->m = m;
        data = new float [n*m];

        for (int j=0; j < n; j++) {
            for (int i=0; i < m; i++) {
                data[i+j*m] = 107. + i + j;
            }
        }
    }

    // operator overloading, used to index 2D array using (i,j)
    float operator()(int i, int j) {
        return this->data[i+j*this->m];
    }

    ~Array2D() //destructor
    {
        delete [] data;
    }

};
```

2D array

Array2D: constructor

~Array2D: destructor

operator(): overload ()

Use of Array2D

```
int main()
{
    Array2D a(10,5);
    printf("a.n = %d\n", a.n);
    printf("a.data[7] = %f\n", a.data[7]);
    printf("a(3,5)= %f\n", a(3,5));
    return 0;
}
```

```
./a.out
a.n = 10
a.data[7] = 110.000000
a(3,5)= 119.000000
```

Use dot operator to access array attributes data and n

Note use of **a(3,5)** like Fortran. Array is continuous in memory and dynamic.

Alternative. Static array a[5][10] (2nd index varies fastest in C/C++/Java). Why? a[5][3] is identical to a(3,5)

We access array as 2D array, although the array is 1D inside the class. The class hides the implementation.

```
    ……

    float operator()(int i, int j) const { //read operation
        return this->data[i+j*this->m];
    }

    float& operator() (int i, int j) { // write operation
        return this->data[i+j*m] ;
    }

    ~Array2D() //destructor
    {
        delete [] data;
    }

};


int main()
{
    Array2D a(10,5);
    printf("a.n = %d\n", a.n);
    printf("a.data[7] = %f\n", a.data[7]);
    printf("a(3,5)= %f\n", a(3,5));
    a(3,5) = 3.14;
    printf("a(3,5)= %f\n", a(3,5));
    return 0;
}
```

```
nagal:L17>ref
a.n = 10
a.data[7] = 110.000000
a(3,5)= 119.000000
a(3,5)= 3.140000
```

2D Array with pointers and references

- Pointers
  - necessary to decrease the amount of data passed to a function through arguments
- References
  - simplify data access inside function when passing data as a reference

Pass by value or pass by refere

```
#include <stdio.h>

class Large {
public:
    float f[10000];
    Large() {
        for (int i=0; i < 10000; i++) {
            f[i] = i*10+7.;
        }
    }
    ~Large() {}
};

void change(Large arr)
{
    arr.f[5] = 20.;
    printf("in function change(): %f\n",arr.f[5]);
    return;
}

void changeR(Large &arr)
{
    arr.f[5] = 20.;
    printf("in function changeR(): %f\n",arr.f[5]);
    return;
}

int main()
{
    Large large;
    printf("before: large[5]= %f\n", large.f[5]);
    change(large);
    printf("after change(): large[5]= %f\n", large.f[5]);
    printf("sizeof(large)= %lu\n", sizeof(large));
    changeR(large);
    printf("after changeR(): large[5]= %f\n", large.f[5]);
    printf("sizeof(large)= %lu\n", sizeof(large));

    return 0;
}
```

```
nagal:L17>t2
before: large[5]= 57.000000
in function change(): 20.000000
after change(): large[5]= 57.000
sizeof(large)= 40000
in function changeR(): 20.000000
after changeR(): large[5]= 20.00
sizeof(large)= 40000
n
```

Pass by Value

- Recall that everything is passed by value (in my interpretation)
  - variables, addresses, references
- If one passes a huge array by value, the computation can slow down substantially
- Demonstrate in class with timer

```
int main() {
    std::clock_t start;
    double duration;
    Large large;
    printf("before: large[5]= %f\n", large.f[5]);
    start = std::clock();
    change(large);
    duration = ( std::clock() - start ) / (double) CLOCKS_PER_SEC;
    printf("after change(): large[5]= %f\n", large.f[5]);
    printf("Duration: %f\n", duration);
    start = std::clock();
    changeR(large);
    duration = ( std::clock() - start ) / (double) CLOCKS_PER_SEC;
    printf("after changeR(): large[5]= %f\n", large.f[5]);
    printf("Duration: %f\n", duration);
}
```

by value

by reference

```
nagal:L17>timer
before: large[5]= 57.000000
in function change(): 20.000000
after change(): large[5]= 57.000000
Duration: 0.001994
in function changeR(): 20.000000
after changeR(): large[5]= 20.000000
Duration: 0.000001
```

```
#include <stdio.h>
#include "array2D.h"

void change(Array2D arr)
{
    arr.data[5] = 270.;     // change value in class
    return;
}

int main()
{
    Array2D ap(3,5);

    printf("ap.data[5] = %f\n", ap.data[5]);
    change(ap);          // pass class as argument
    printf("ap.data[5] = %f\n", ap.data[5]);
    return 0;
}
```

Class is passed by value

CRASH

```
nagal:L17>clang++ test3.cpp -o t3
nagal:L17>t3
ap.data[5] = 108.000000
ap.data[5] = 270.000000
t3(49654,0x7fff7c0e0300) malloc: *** error for object 0x7fca69404dd0: pointer
being freed was not allocated
*** set a breakpoint in malloc_error_break to debug
Abort trap: 6
```

Why the error message

- We must analyze the destructor
- Rewrite the class with diagnostics
- Recall that the class has a **float* data**
  - **see next slide**
- The class is passed by value, so is copied
- Therefore, **data** is copied
- However, **the contents of data are not copied!!**

Simple class

```
class Array2D
{
public:
    int n, m;
    float* data;
    Array2D(int n, int m)
    {
        printf("inside constructor\n");
        this->n = n;
        this->m = m;
        data = new float [n*m];

        for (int j=0; j < m; j++) {
            for (int i=0; i < n; i++) {
                data[i+j*m] = 107. + i + j;
            }
        }
    }
    float operator() (int i, int j) const { // read operation
        return this->data[i+j*m] ;
    }
    float& operator() (int i, int j) { // write operation
        return this->data[i+j*m] ;
    }
    ~Array2D() //destructor
    {
        delete [] data;
    }
};
```

```
int main()
{
    Array2D ap(3,5);

    printf("ap.data[5] = %f\n", ap.data[5]);
    change(ap);         // pass class as argument
    printf("ap.data[5] = %f\n", ap.data[5]);
    return 0;
}
```

Constructor called
Default copy constructor called (argument is copied)
by default, all attributes are copied: n,m, data, but NOT the dynamically generated array!!
Before exiting change, the destructor is called
Therefore, data is freed: delete [] data;
Upon from main, Array2D is freed, and so data is freed again, leading to an error.
Fix: create a copy constructor

Copy Constructor

```
Array2D(Array2D& arr) {// Note the use of a reference!!!
    printf("inside copy constructor\n");
    this->n = arr.n;
    this->m = arr.m;
    this->data = new float [n*m];
    for (int i=0; i < n*m; i++) {
        this->data[i] = arr.data[i]; // deep copy
    }
}
```

Constructor

```
Array2D(int n, int m)
{
    printf("inside constructor\n");
    this->n = n;
    this->m = m;
    data = new float [n*m];

    for (int j=0; j < m; j++) {
        for (int i=0; i < n; i++) {
            data[i+j*m] = 107. + i + j;
        }
    }
}
```

Copy Constructor

```
Array2D(Array2D& arr) {// Note the use of a reference!!!
    printf("inside copy constructor\n");
    this->n = arr.n;
    this->m = arr.m;
    this->data = new float [n*m];
    for (int i=0; i < n*m; i++) {
        this->data[i] = arr.data[i]; // deep copy
    }
}
```

```
int main()
{
    Array2D ap(3,5);

    printf("ap.data[5] = %f\n", ap.data[5]);
    change(ap);
    printf("ap.data[5] = %f\n", ap.data[5]);
    return 0;
}
```

```
nagal:L17>t3
inside standard constructor
ap.data[5] = 1.000000
inside copy constructor
inside destructor
ap.data[5] = 1.000000
inside destructor
```