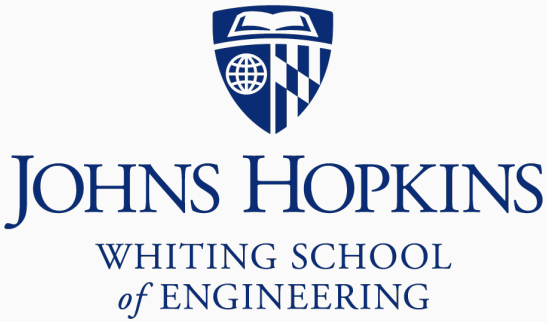
References

Ben Langmead

ben.langmead@gmail.com

www.langmead-lab.org



Source markdown available at github.com/BenLangmead/c-cpp-notes

References

Like iterators, references provide pointer-like functionality without many of the pointer drawbacks

Reference variable is an *alias*, another name for an existing variable

Restrictions make them safer than pointers:

- Can't be NULL
- Must be initialized immediately
- Once set to alias a variable, cannot later alias a different variable

Because of these, it's harder (though still possible) to access memory that doesn't belong to you via reference

C++: References

A *reference* to variable of type `int` has type `int&`

- `&` is *part of the type*
- Reminds us of "address of" operator `&`

C++: References

Example with pointer:

```
#include <iostream>
using std::cout;  using std::endl;

int main() {
    int i = 1;
    int *j = &i;
    cout << "i=" << i << ", *j=" << *j << endl;

    *j = 9;
    cout << "i=" << i << ", *j=" << *j << endl;
    return 0;
}
```

```
$ g++ -c ref_ptr_1.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o ref_ptr_1 ref_ptr_1.o
$ ./ref_ptr_1
i=1, *j=1
i=9, *j=9
```

C++: References

Same example with reference:

```
#include <iostream>
using std::cout;  using std::endl;

int main() {
    int i = 1;
    int& j = i; // j is an alias (another name) for i
    cout << "i=" << i << ", j=" << j << endl;

    j = 9; // no dereference (*) needed
    cout << "i=" << i << ", j=" << j << endl;
    return 0;
}
```

```
$ g++ -c ref_ptr_2.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o ref_ptr_2 ref_ptr_2.o
$ ./ref_ptr_2
i=1, j=1
i=9, j=9
```

C++: References

You can get the address of a reference variable with &; it has the same address as the variable it aliases

```
#include <iostream>
using std::cout;  using std::endl;

int main() {
    int a = 5;
    int& b = a;
    cout << "&a=" << &a << endl << "&b=" << &b << endl;
    return 0;
}
```

```
$ g++ -c ref_addr.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o ref_addr ref_addr.o
$ ./ref_addr
&a=0x7ffeb7dacc94
&b=0x7ffeb7dacc94
```

C++: References

Function parameters with reference type are passed "by reference"

```
// if you have int a = 1, b = 2;
// then call like this: swap(a, b) -- no &s!
void swap(int& a, int& b) {
    int tmp = a;
    a = b;
    b = tmp;
}
```

Passing by reference is like passing by pointer

- Callee can modify variable in caller
- Large data structures can be passed without copying

C++: References

```
#include <iostream>
using std::cout;  using std::endl;

void swap(int& a, int& b) {
    int tmp = a;
    a = b;
    b = tmp;
}

int main() {
    int a = 1, b = 9;
    swap(a, b);
    cout << "a=" << a << ", b=" << b << endl;
    return 0;
}
```

```
$ g++ -c ref_swap.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o ref_swap ref_swap.o
$ ./ref_swap
a=9, b=1
```

C++: References

Recall this example; ch passed by reference to `cin.get(char&)`

```
#include <iostream>
#include <cctype>
using std::cout;  using std::endl;

int main() {
    char ch;
    // read standard input char by char
    while(cin.get(ch)) { // pass ch by reference!
        cout << toupper(ch);
    }
    cout << endl;
    return 0;
}
```

C++: References

C++ has *both* pass-by-value (non-reference parameters) *and* pass-by-reference (reference parameters)

Function can have a mix of pass-by-value and pass-by-reference parameters

C++: References

```cpp
#include <iostream>
using std::cout;  using std::endl;

// `int a` and `int b` are passed *by value*
// `int& quo` and `int& rem` are passed *by reference*
void divmod(int a, int b, int& quo, int& rem) {
    quo = a / b;
    rem = a % b;
}

int main() {
    int a = 10, b = 3, quo, rem;
    divmod(a, b, quo, rem);
    cout << "a=" << a << ", b=" << b
         << ", quo=" << quo << ", rem=" << rem << endl;
    return 0;
}
```

C++: References

```
$ g++ -c divmod_ref.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o divmod_ref divmod_ref.o
$ ./divmod_ref
a=10, b=3, quo=3, rem=1
```

C++: References

Looking at the function call doesn't tell you which parameters are passed by value or by reference:

```
divmod(a, b, quo, rem); // ???
```

You have to look at the callee's parameter types:

```
void divmod(int a, int b, int& quo, int& rem) {
    ...
}
```

C++: References

We can return a reference

```
#include <iostream>
using std::cout;  using std::endl;

int& minref(int& a, int& b) {
    if(a < b) {
        return a;
    } else {
        return b;
    }
}

int main() {
    int a = 5, b = 10;
    int& min = minref(a, b);
    min = 12;
    cout << "a=" << a << ", b=" << b << ", min=" << min << endl;
    return 0;
}
```

C++: References

```
$ g++ -c ref_min.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o ref_min ref_min.o
$ ./ref_min
a=12, b=10, min=12
```

`minref` returns a reference to a

`min = 12` modifies both `min` and a

What if we make `minref`'s arguments non-references?

C++: References

```cpp
#include <iostream>
using std::cout;  using std::endl;

int& minref(int a, int b) {
    if(a < b) {
        return a;
    } else {
        return b;
    }
}

int main() {
    int a = 5, b = 10;
    int& min = minref(a, b);
    min = 6;
    cout << "a=" << a << ", b=" << b << ", min=" << min << endl;
    return 0;
}
```

C++: References

```
$ g++ -c %PREV% -std=c++11 -pedantic -Wall -Wextra
ref_min_2.cpp: In function 'int& minref(int, int)':
ref_min_2.cpp:4:17: warning: reference to local variable 'a' returned
  int& minref(int a, int b) {
                 ^
ref_min_2.cpp:4:24: warning: reference to local variable 'b' returned
 int& minref(int a, int b) {
                        ^
```

C++: References

Returning a reference to a local variable is just as bad as returning a pointer to one. In our original `minref` function, this was OK because the parameters themselves were references.

```
int& minref(int& a, int& b) {
    if(a < b) {
        return a;
    } else {
        return b;
    }
}
```

C++: References

Once a reference is set to alias a variable, it cannot later be set to alias another variable

Example. . .

C++: References

```
#include <iostream>
using std::cout;  using std::endl;

int main() {
    int a = 5, b = 10;
    int& c = a;
    cout << "a=" << a << ", c=" << c << endl;
    c = b; // this *doesn't* change c to alias b
           // instead, sets c (and a) to 10
    cout << "a=" << a << ", c=" << c << endl;
    return 0;
}
```

```
$ g++ -c ref_reset.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o ref_reset ref_reset.o
$ ./ref_reset
a=5, c=5
a=10, c=10
```

C++: References

A reference variable must be initialized immediately

```
#include <iostream>
using std::cout;  using std::endl;

int main() {
    int& a; // won't compile
    int b = 10;
    a = b;
    cout << a << endl;
    return 0;
}
```

```
$ g++ -c ref_delay.cpp -std=c++11 -pedantic -Wall -Wextra
ref_delay.cpp: In function 'int main()':
ref_delay.cpp:5:10: error: 'a' declared as reference but not initialized
     int& a; // won't compile
          ^
```

C++: References

A reference cannot be NULL

```
#include <iostream>
using std::cout;  using std::endl;

int main() {
    int& a = NULL;
    if(a == NULL) {
        cout << "a is NULL" << endl;
    }
    return 0;
}
```

C++: References

```
$ g++ -c ref_null.cpp -std=c++11 -pedantic -Wall -Wextra
ref_null.cpp: In function 'int main()':
ref_null.cpp:5:14: warning: converting to non-pointer type 'int' from NULL
[-Wconversion-null]
     int& a = NULL;
              ^~~~
In file included from /usr/include/_G_config.h:15:0,
                 from /usr/include/libio.h:31,
                 from /usr/include/stdio.h:41,
                 from /usr/include/c++/7/cstdio:42,
                 from /usr/include/c++/7/ext/string_conversions.h:43,
                 from /usr/include/c++/7/bits/basic_string.h:6347,
                 from /usr/include/c++/7/string:52,
                 from /usr/include/c++/7/bits/locale_classes.h:40,
                 from /usr/include/c++/7/bits/ios_base.h:41,
                 from /usr/include/c++/7/ios:42,
                 from /usr/include/c++/7/ostream:38,
                 from /usr/include/c++/7/iostream:39,
                 from ref_null.cpp:1:
ref_null.cpp:5:14: error: cannot bind non-const lvalue reference of type 'int&'
to an rvalue of type 'int'
     int& a = NULL;
              ^
ref_null.cpp:6:13: warning: NULL used in arithmetic [-Wpointer-arith]
     if(a == NULL) {
             ^~~~
```

C++: References

A reference can be `const`

- `const` reference cannot be modified after initialization

You may still be able to modify the original variable

- . . . if it's non-`const`, or
- . . . via a non-`const` reference to the same variable

C++: References

```cpp
#include <iostream>
using std::cout;  using std::endl;

int main() {
    int a = 1;
    int& b = a;
    const int& c = a;
    a = 2;
    cout << "a=" << a << ", b=" << b << ", c=" << c << endl;
    b = 3; // this is fine
    cout << "a=" << a << ", b=" << b << ", c=" << c << endl;
    c = 4; // *** not allowed ***
    cout << "a=" << a << ", b=" << b << ", c=" << c << endl;
    return 0;
}
```

C++: References

```
$ g++ -c ref_const.cpp -std=c++11 -pedantic -Wall -Wextra
ref_const.cpp: In function 'int main()':
ref_const.cpp:12:9: error: assignment of read-only reference 'c'
     c = 4; // *** not allowed ***
         ^
```