COMP421 Bootcamp

Ben Berg, Zhongrui (reads John-Ray) Chen

Department of Computer Science, University of North Carolina at Chapel Hill

Aug 25, 2025

Plans for the day

- This bootcamp assumes passing familiarity with C and Java.
- We will go over some basic C++ syntax and features.
- Lastly, get your hands dirty on simple C++ tasks.
- Enjoy your food and let’s get started!

What is C++?

- You learned Java in COMP 301
- You learned C in COMP 211/311
- C++:
  - ✓ Object-oriented programming from Java
  - ✓ Pointers and efficiency from C
  - ✓ A lot more to offer…

C++: Basic Syntax

Return type of the function

```
void changeName(Person p) {
    p.setName("B");
}
```

Main function

Instantiating an object

```
int main() {
    Person p("A", 10);
    changeName(p);
    return 0;
}
```

Returns 0 if code finishes without error

Common pitfalls / subtle differences

- Values, references and pointers
- Objects and inheritance
- Threads and locks

C++: What’s different?

Difference 1: Reference Types

Passing by copying

```
void changeName(Person p) {
   p.setName("B");
}
int main() {
   Person p("A", 10);
   changeName(p);
   std::cout << p.getName() << std::endl;
   // prints "A"
   return 0;
}
```

In Java: only references are passed around

Passing by reference

```
void changeName(Person &p) {
   p.setName("B");
}
int main() {
   Person p("A", 10);
   changeName(p);
   std::cout << p.getName() << std::endl;
   // prints "B"
   return 0;
}
```

In C++: passing reference using Type&

In C: there is no reference type. Pass by pointers (Person*).

C++: What’s different?

Difference 2: Polymorphism

```
class B : public A {
public:
   B(int num) : num_(num) {}
   void print() override {
       std::cout << "B " << num_ << std::endl;
   }
private:
   int num_;
};
int main() {
   B b = B(1);
   A a = b;
   a.print(); // prints "A"
   b.print(); // prints "B 1"
   return 0;
}
```

```
class A {
public:
   void print() {
       std::cout << "A" << std::endl;
   }
};
```

C++: Upcasting works differently from Java. It slices the object.

C++: What’s different?

Difference 2: Polymorphism

```
class A {
public:
   void print() {
       std::cout << "A" << std::endl;
   }
};
class B : public A {
public:
   void print() {
       std::cout << "B" << std::endl;
   }
};
int main() {
   B b = B();
   A &a = b;
   a.print(); // prints "A"
}
```

In Java: any non-static method call is a *dynamic dispatch call*

In C++: unless specified otherwise, the compiler decides which function to call beforehand.

C: what is object-oriented design?

Virtual methods in C++

Difference 2: Polymorphism

```
class A {
public:
    virtual void print() {
        std::cout << "A" << std::endl;
    }
};
class B : public A {
public:
    void print() override {
        std::cout << "B" << std::endl;
    }
};
int main() {
    B b = B();
    A &a = b;
    a.print(); // prints "B"
}
```

In C++: use ”virtual” keyword to specify dynamic dispatch.

Object-oriented C++: constructor and destructor

C++ Constructor

Initializer list

```
B(int num, std::string name) : num_(num), name_(name) {}
```

When is the constructor called?

```
Person a("A", 20);
```

C++ Destructor

```
~B() {
   delete …;
}
```

When is the destructor called?

Why don’t we have to worry about this in Java?

Depends on the lifetime of the object.

Difference 3: C++ Object Lifetime

Java: automatically managed lifetime with garbage collection.

Compiler-managed lifetime

(Stack, AKA automatic storage duration)

```
{
   Person a("A", 20);
}
```

Object a exists only inside the braces

Person b = a; ⇔ Person b = Person(a);

Manually-managed lifetime

(Heap, AKA dynamic storage duration)

```
{
   Person *pa = new Person("A", 20);
}
```

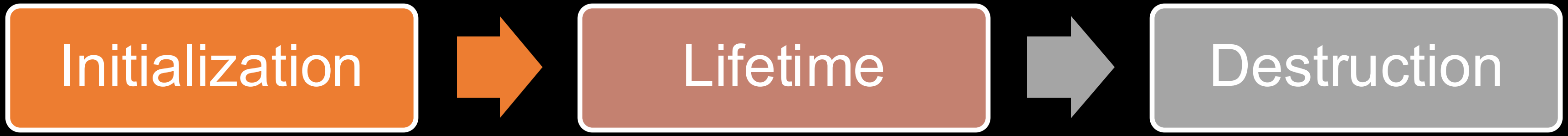
Object is accessible outside of the braces

Person *pb = pa assigns a memory address

C++ Object Lifetimes

```
class Person {
public:
    Person(std::string name, int age) : name_(name), age_(age) {}
    ~Person() {
        std::cout << "destructor called" << std::endl;
    }
private:
    std::string name_;
    int age_;
};
```

Heap allocated	`new Person("A", 20);`	**Live until deleted**	`~Person()` **called on deletion**
Stack allocated	`Person a("A", 20);`	**Live until out of scope**	`~Person()` **called when out of scope**



See objects.cpp in bsb20/421-bootcamp for more examples

What’s wrong with C?

```
void memory_leak_function() {
    int *ptr = (int *) malloc(sizeof(int));
    *ptr = 10;
}
int main() {
    for (int i = 0; i < 1000; i++) memory_leak_function();
    int *a = (int*) malloc(sizeof(int) * 5);
    int *b = a;
    free(a);
    return 0;
}
```

*ptr not accessible outside of this function

Issue: the memory it points to is not getting deallocated

b points to deallocated memory! Undefined behavior.

Original C++ solution: new and delete with object destructors 🤢

Modern C++ solution: STL containers (today) and smart pointers (next)

C++ Arrays (in containers)

C++

Java

Containers deallocate the memory on their destructors

C++ unordered_map

```
std::unordered_map<std::string, int> student_grades;
student_grades["B"] = 101;
std::cout << student_grades["B"] << std::endl;
```

insert one/more mappings into the map

```
student_grades.insert({{"E", 103}, {"F", 104}, {"G", 105}});
```

how to tell if a key is in the map

```
if (student_grades.count("C") == 0) {
    std::cout << "No student named C" << std::endl;
}
```

Iterating over an unordered_map

```
for (auto &pair : student_grades) {
    std::cout << pair.first << " " << pair.second << std::endl;
}
```

Task

Given a string of words separated by a single space, count word frequencies.

See the following example

- Example Input:
  - the quick brown fox jumps over the lazy dog the quick fox
- Example Output:

```
Word Frequencies:
"lazy": 1
"jumps": 1
"dog": 1
"the": 3
"fox": 2
"brown": 1
"over": 1
"quick": 2
```

Threads

- Parallel execution units that shares memory
- Next: Threads in C++

Threads without locks

```
int count = 0;
std::mutex m;
void add_count() {
   count += 1;
}
int main() {
   std::thread t1(add_count);
   std::thread t2(add_count);
   t1.join();
   t2.join();
   std::cout << "Printing count: " << count << std::endl;
   return 0;
}
```

What is the output?

Possible scenario:

Thread t1 and t2 reads count as 0 at the same time.

Thread t1 and t2 trying to set count to 1 at the same time.

count becomes 1 after execution when we want 2.

Thread Synchronization

```
int count = 0;
std::mutex m;
void add_count() {
    {
        std::scoped_lock lock(m);
        count += 1;
    }
}
int main() {
    std::thread t1(add_count);
    std::thread t2(add_count);
    t1.join();
    t2.join();
    std::cout << "Printing count: " << count << std::endl;
    return 0;
}
```

Solution: mutual exclusion lock

```
class scoped_lock {
    scoped_lock(std::mutex &m) : m_(m) {
        m.lock();
    }
    ~scoped_lock() {
        m_.unlock();
    }
    std::mutex& m_;
};
```