Robotics Corner

Robotics Corner

Smart Pointers

Smart pointers were provided in C++11 to solve the problem of raw pointers.

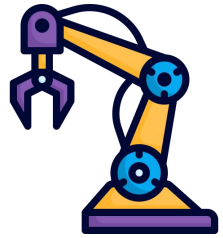
01 The problem: raw pointers

02 The solution: smart pointers

03 How to implement Smart pointer

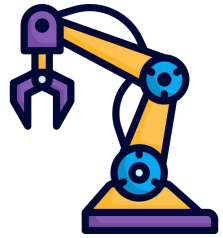
04 Types of Smart pointers

ROBOTICS CORNER



Why Smart Pointers?

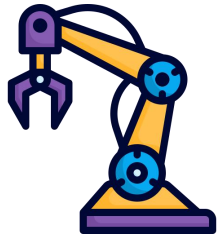
- When to delete an object?
  - No deletion → **memory leaks**
  - Early deletion (others still pointing to) → **dangling pointers**
  - **Double-freeing**



The good old pointer

```
void oldPointer(){
   Foo * myPtr = new Foo();
   myPtr->method();
}
```

Memory leak



The good Old pointer

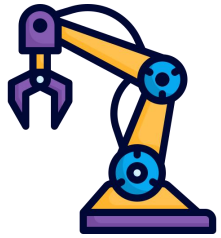
```
void oldPointer1(){
   Foo * myPtr = new Foo();
   myPtr->method();
}
```

Memory leak

```
void oldPointer2(){
   Foo * myPtr = new Foo();
   myPtr->method();
   delete myPtr;
}
```

Could cause memory leak When?

ROBOTICS CORNER



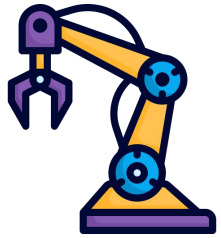
The Old and the New

```
void oldPointer(){
   Foo * myPtr = new Foo();
   myPtr->method();
}
```

Memory leak

```
void newPointer(){
   shared_ptr<Foo> myPtr (new Foo());
   myPtr->method();
}
```

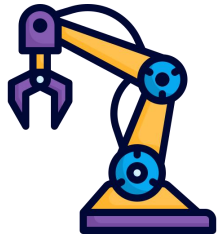
The solution: smart pointers



- Behave like built-in (raw) pointers
- Also manage dynamically created objects
  - Objects get deleted in smart pointer destructor

- Type of ownership:
  - unique
  - Shared
  - Weak

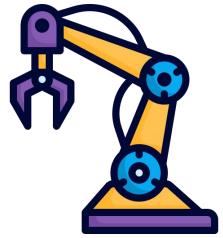
03 How to implement Smart pointer



```
void newPointer(){
   shared_ptr<Foo> myPtr (new Foo());
   myPtr->method();
}
```

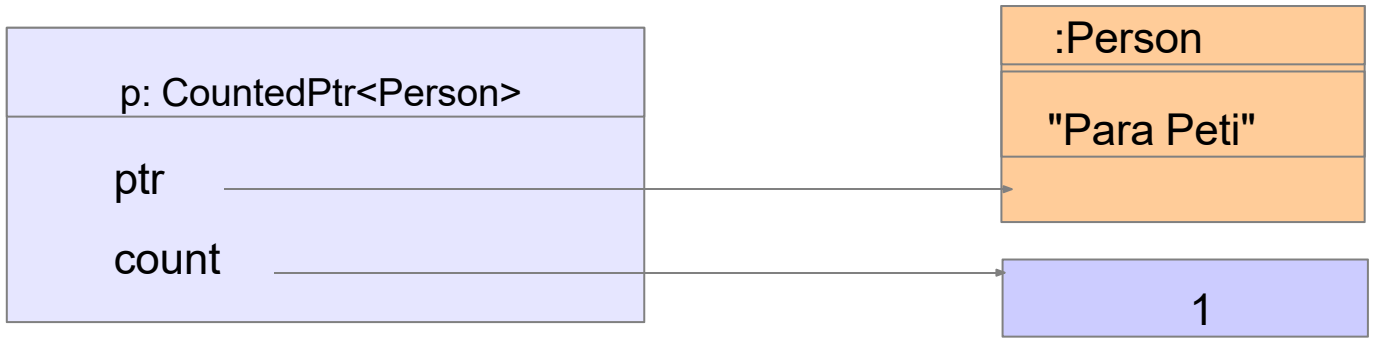
```
void newPointer(){
   auto myPtr = make_shared<Foo>();
   myPtr->method();
}
```

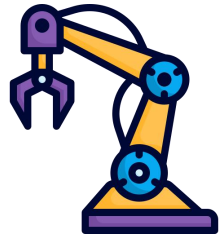
Static
factory method



Implementing your own smart pointer class

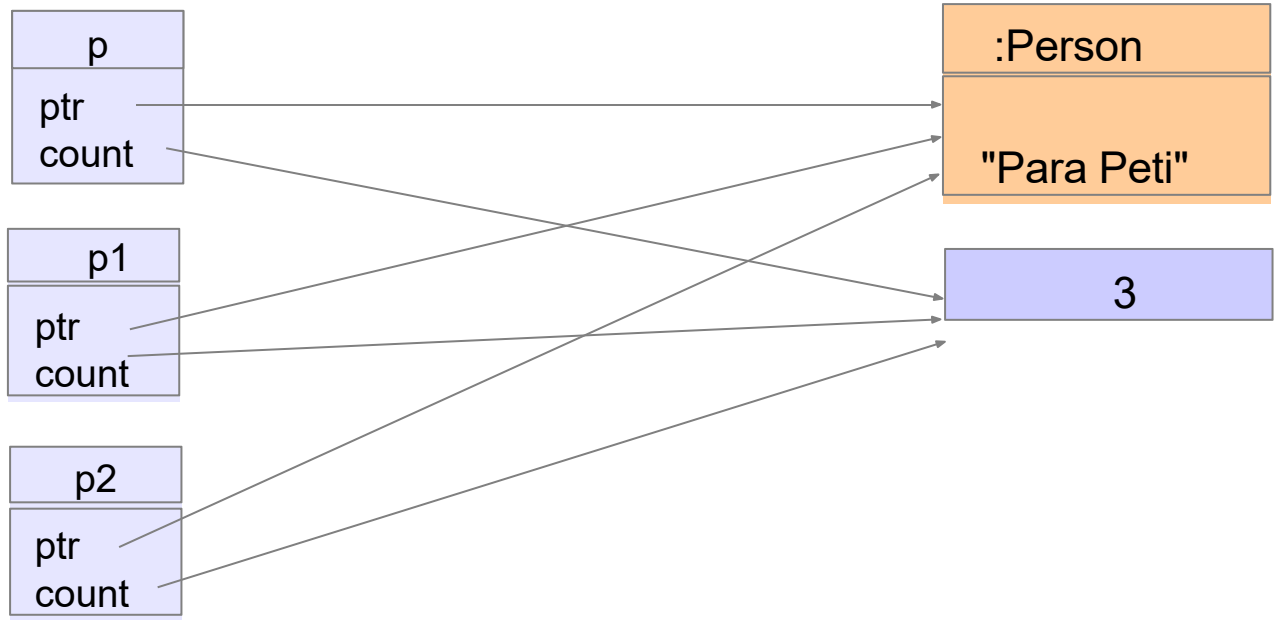
```
CountedPtr<Person> p(new Person("Para Peti",1980));
```

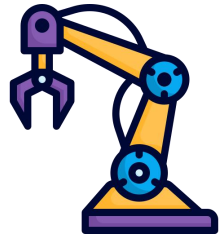




Implementing your own smart pointer class

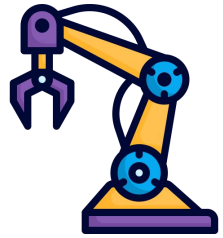
```
CountedPtr<Person> p1 = p;
CountedPtr<Person> p2 = p;
```





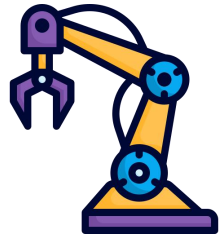
Implementation (1)

```
template < class T>
class CountedPtr{
  T * ptr;
  long * count;
public:
   ...
};
```



Implementation (2)

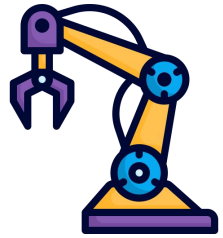
```
CountedPtr( T * p = 0 ):ptr( p ),
    count( new long(1)){
}

CountedPtr( const CountedPtr<T>& p ): ptr( p.ptr),
    count(p.count){
    ++(*count);
}

~CountedPtr(){
   --(*count);
   if( *count == 0 ){
     delete count; delete ptr;
   }
}
```



Implementation (3)

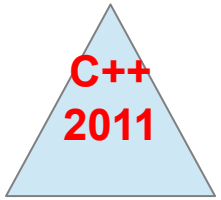
```
CountedPtr<T>& operator=( const CountedPtr<T>&
   p ){ if( this != &p ){
      --(*count);
      if( *count == 0 ){ delete count; delete ptr; }
      this->ptr = p.ptr;
      this->count = p.count;
      ++(*count);
   }
   return *this;
}

T& operator*()  const{ return *ptr;}

T* operator->() const{ return  ptr;}
```

Smart Pointer Types

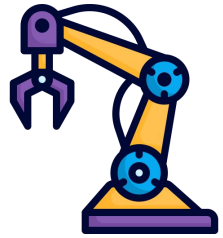


- `unique_ptr`
- `shared_ptr`
- `weak_ptr`

```
#include <memory>
```

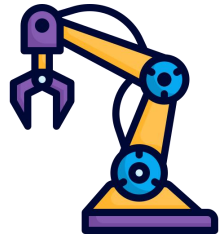


It is recommended to use smart pointers!



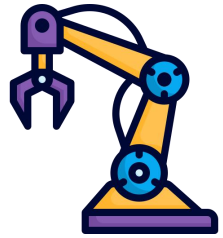
unique_ptr

- it will automatically free the resource in case of the unique_ptr goes out of scope.

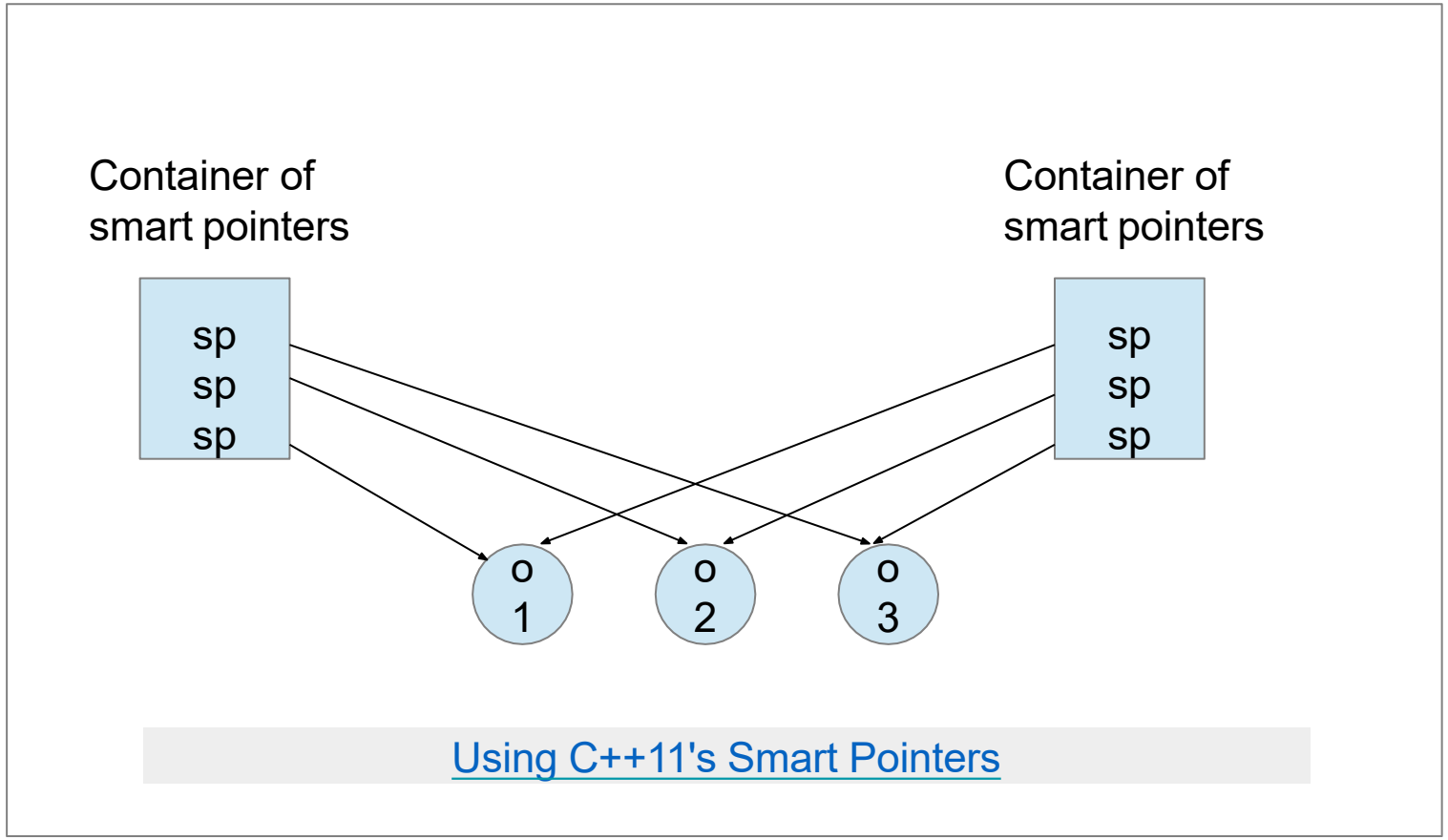


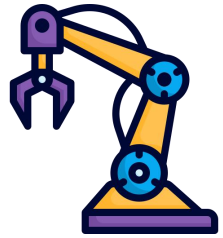
shared_ptr

- Each time a `shared_ptr` is assigned
  - a **reference count** is incremented (there is one more “owner” of the data)
    - When a `shared_ptr` goes out of scope
      - the **reference count** is decremented
  - if **reference_count = 0** the object referenced by the pointer is freed.

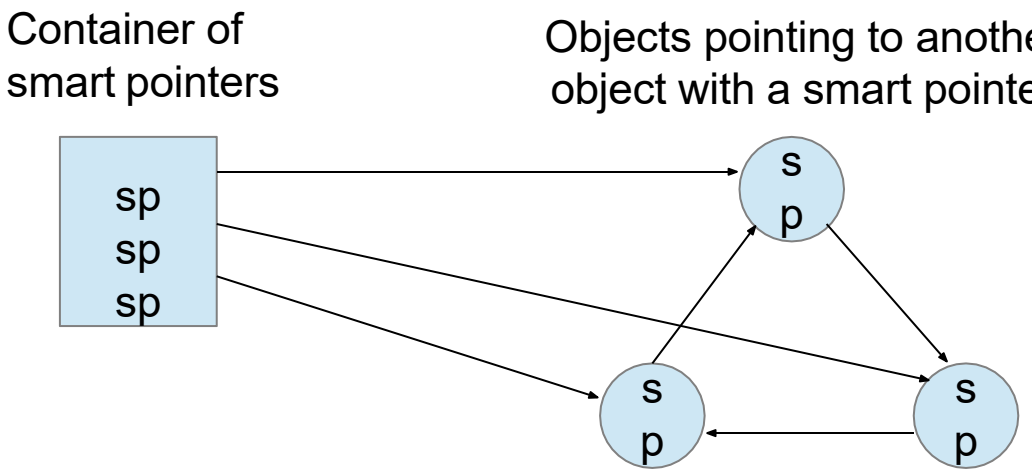


Shared ownership with `shared_ptr`

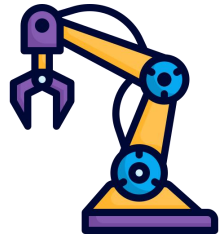




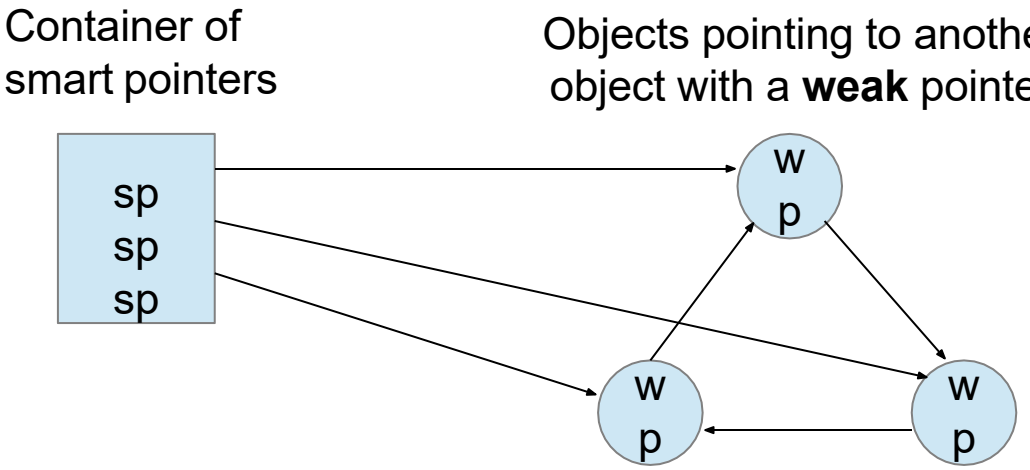
Problem with `shared_ptr`



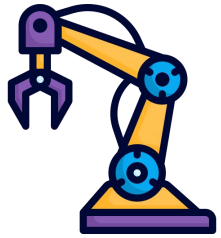
Using C++11's Smart Pointers



Solution: `weak_ptr`



Using C++11's Smart Pointers

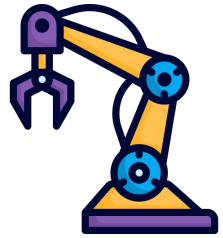


weak_ptr

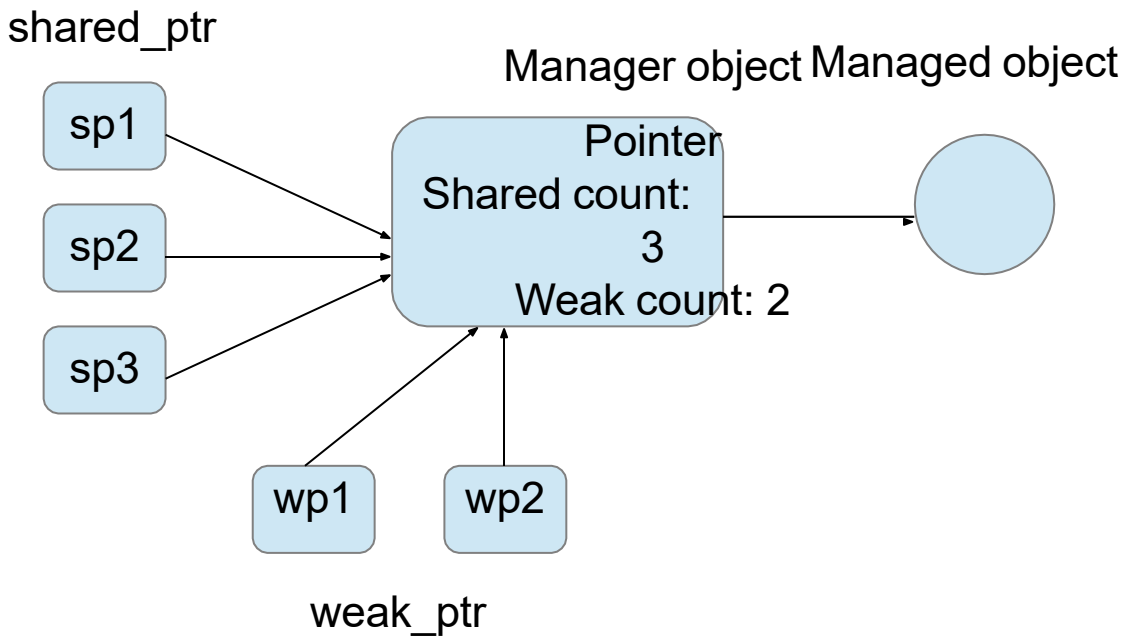
- Observe an object, but does not influence its lifetime
- Like raw pointers - the weak pointers do not keep the pointed object alive
- Unlike raw pointers – the weak pointers know about the existence of pointed-to object

Using C++11's Smart Pointers

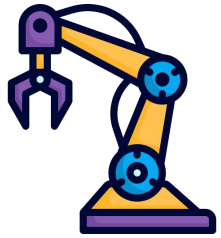
ROBOTICS CORNER



How smart pointers work



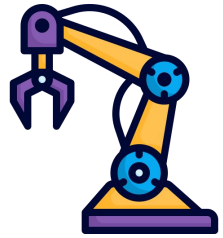
Using C++11's Smart Pointers



Restrictions in using smart pointers

- Can be used to refer to objects allocated with `new` (can be deleted with `delete`).
- Avoid using raw pointer to the object referred by a smart pointer.

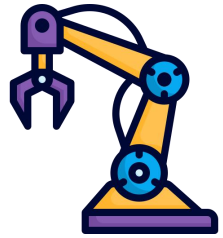
Using C++11's Smart Pointers



Inheritance and shared_ptr

```
void greeting( shared_ptr<Person>&
  ptr ){  cout<<"Hello "<<(ptr.get())-
  >getFname()<<" "
                  <<(ptr.get())->getLname()<<endl;
}

int main(int argc, char** argv) {
  shared_ptr<Person> ptr_person(new Person("John","Smith"));
  cout<<*ptr_person<<endl;
  greeting( ptr_person );

  shared_ptr<Manager> ptr_manager(new Manager("Black","Smith", "IT"));
  cout<<*ptr_manager<<endl;
  ptr_person = ptr_manager;
  cout<<*ptr_person<<endl;
  return 0;
}
```

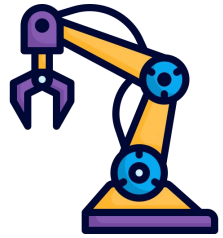
ROBOTICS CORNER



unique_ptr usage

```
// p owns the Person
unique_ptr<Person> uptr(new Person("Mary", "Brown"));

unique_ptr<Person> uptr1( uptr ); //ERROR – Compile time

unique_ptr<Person> uptr2;          //OK. Empty unique_ptr

uptr2 = uptr1;                      //ERROR – Compile time
uptr2 = move( uptr );                //OK. uptr2 is the owner
cout<<"uptr2: "<<*uptr2<<endl;      //OK
cout<<"uptr : "<<*uptr <<endl;      //ERROR – Run time

unique_ptr<Person> uptr3 = make_unique<Person>("John","Dee");
cout<<*uptr3<<endl;
```

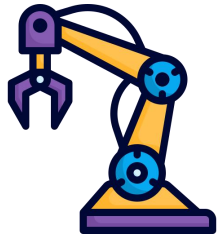
Static
Factory Method



unique_ptr usage (2)

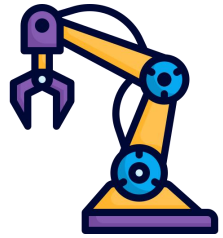
```
unique_ptr<Person> uptr1 =
                            make_unique<Person>("Mary","Black");
unique_ptr<Person> uptr2 = make_unique<Person>("John","Dee");
cout<<*uptr2<<endl;

vector<unique_ptr<Person> > vec;
vec.push_back( uptr1  );
vec.push_back( uptr2 );

cout<<"Vec [";
for( auto e:
    vec )
    { cout<<*e<<"
    ";
}
cout<<"]"<<endl;
```

Find the errors and correct them!!!



unique_ptr usage (2)

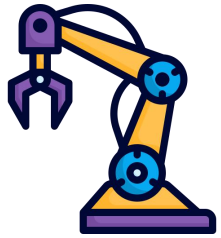
```
unique_ptr<Person> uptr1 =
                            make_unique<Person>("Mary","Black");
unique_ptr<Person> uptr2 = make_unique<Person>("John","Dee");
cout<<*uptr2<<endl;

vector<unique_ptr<Person> > vec;
vec.push_back( move( uptr1 ) );
vec.push_back( move( uptr2 ) );

cout<<"Vec [";
for( auto& e:
    vec )
    { cout<<*e<<" ";
}
cout<<"]"<<endl;
```



Thank You

Do you have any questions?

ROBOTICS CORNER



ROBOTICS CORNER

01211626904

www.roboticscorner.tech

Robotics Corner

Robotics Corner

Robotics Corner

Robotics Corner