

# ECE 220

## Lecture x001B - Final exam review

# About last time

# About last time

- Review templated tree code

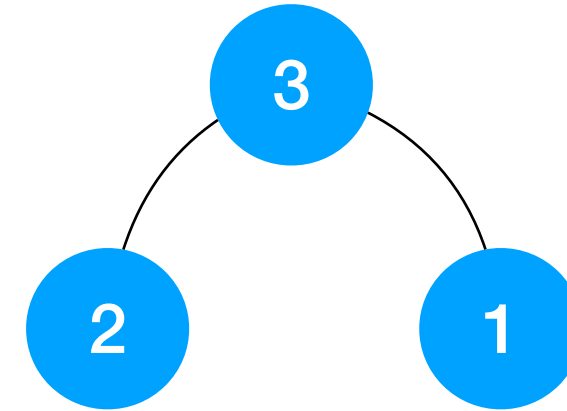
# About last time

- Review templated tree code
- Review isBST code

# About last time

- Review templated tree code
- Review isBST code
- Convert to doubly-linked-list

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

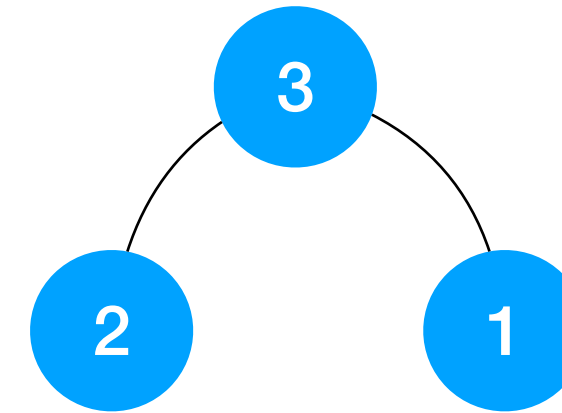
    prev = cursor;
    bool right = is_bst(cursor->right, prev);

    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

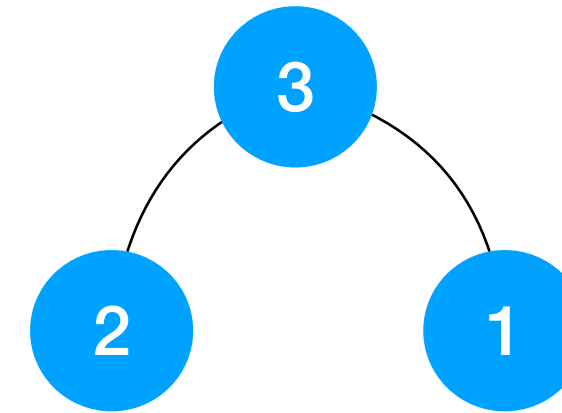
    prev = cursor;
    bool right = is_bst(cursor->right, prev);

    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

# Is this a BST?



```
→ bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

    return (left && right);
}
```

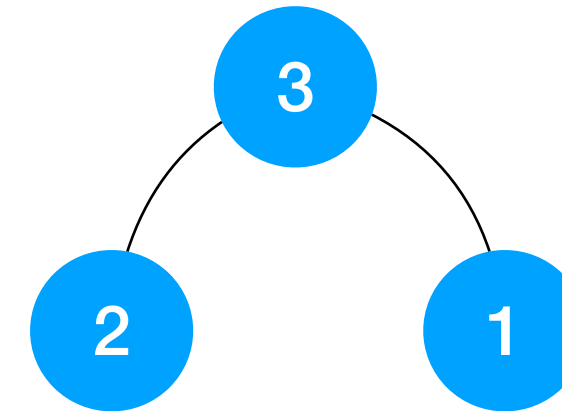
```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
--------	------	------	-------



# Is this a BST?



```
→ bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

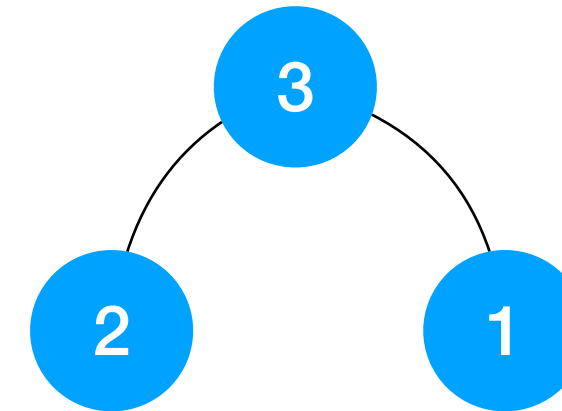
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		

# Is this a BST?



```
→ bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

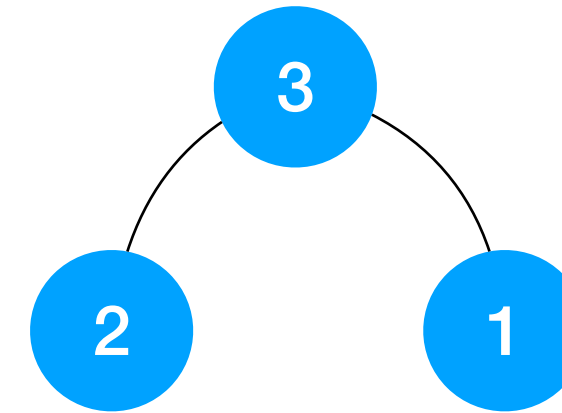
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;
    → bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

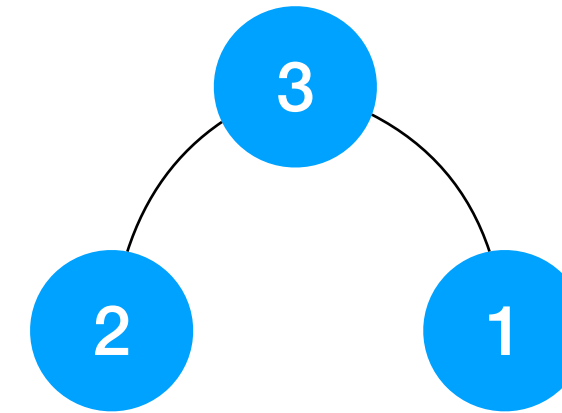
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		

# Is this a BST?



```
→ bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

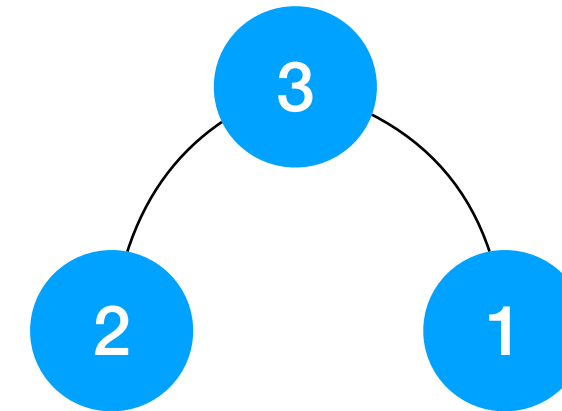
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		
2	NULL		

# Is this a BST?



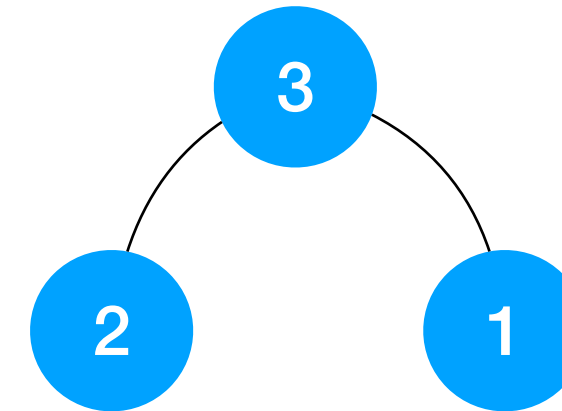
```
bool is_bst(node *cursor, node *&prev){  
    if (cursor==NULL)  
        return true;  
  
    bool left = is_bst(cursor->left, prev);  
  
    if (prev!=NULL && cursor->data <= prev->data)  
        return false;  
  
    prev = cursor;  
    bool right = is_bst(cursor->right, prev);  
  
    return (left && right);  
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))  
    cout << "Tree is BST";  
else  
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		
2	NULL		

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;
    → bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

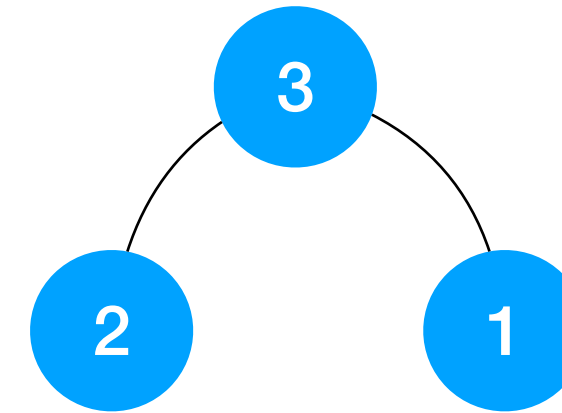
    return (left && right);
}
```

```
node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		
2	NULL		

# Is this a BST?



```
→ bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

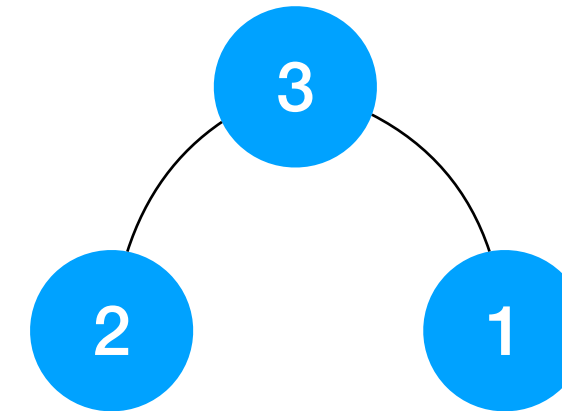
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		
2	NULL		
NULL			

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){  
    if (cursor==NULL)  
        return true;  
  
    bool left = is_bst(cursor->left, prev);  
  
    if (prev!=NULL && cursor->data <= prev->data)  
        return false;  
  
    prev = cursor;  
    bool right = is_bst(cursor->right, prev);  
  
    return (left && right);  
}
```

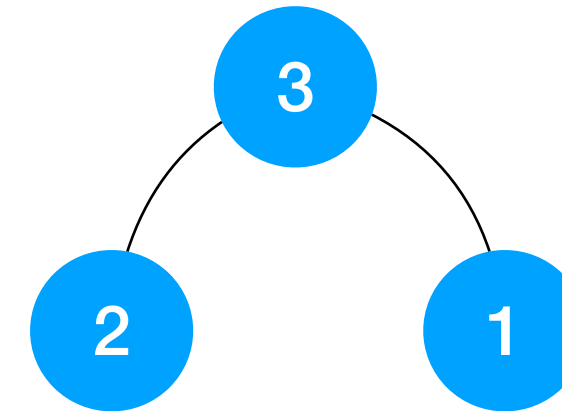
```
node *prev = NULL;
```

```
if (is_bst(root, prev))  
    cout << "Tree is BST";  
else  
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		
2	NULL		
NULL			



# Is this a BST?

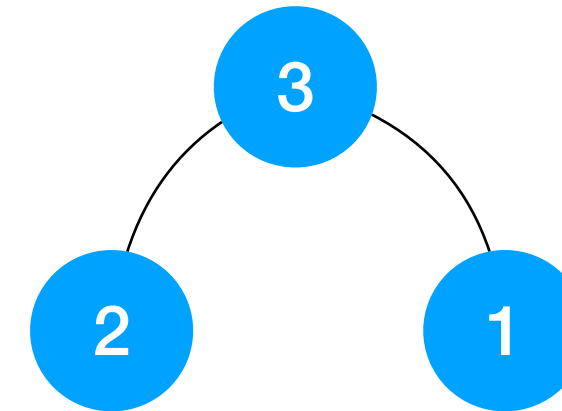


```
bool is_bst(node *cursor, node *&prev){  
    if (cursor==NULL)  
        return true;  
    → bool left = is_bst(cursor->left, prev);  
  
    if (prev!=NULL && cursor->data <= prev->data)  
        return false;  
  
    prev = cursor;  
    bool right = is_bst(cursor->right, prev);  
  
    return (left && right);  
}
```

```
node *prev = NULL;  
  
if (is_bst(root, prev))  
    cout << "Tree is BST";  
else  
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		
2	NULL		
NULL			

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;
    → bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

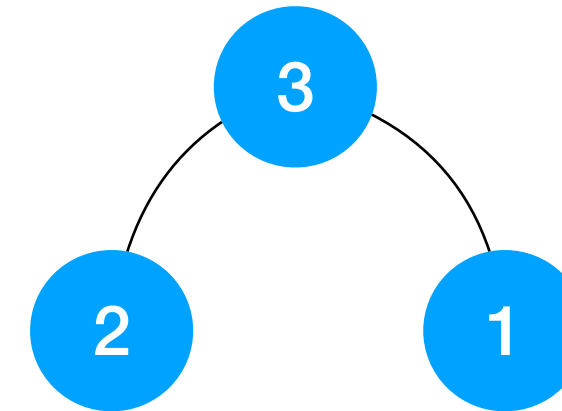
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		
2	NULL		
NULL			

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;
    → bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

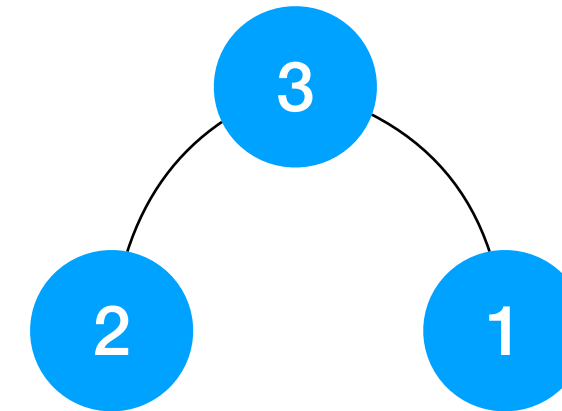
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		
2	NULL	true	
NULL			

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    → if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

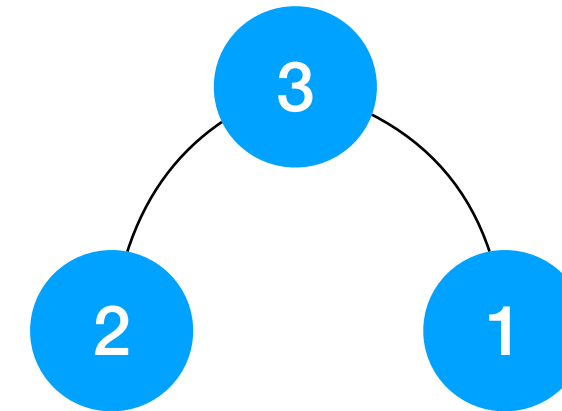
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		
2	NULL	true	
NULL			

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    → prev = cursor;
    bool right = is_bst(cursor->right, prev);

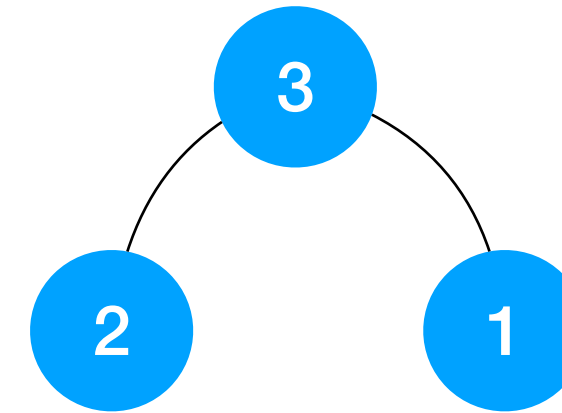
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	NULL		
2	NULL	true	
1			
NULL			

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    → prev = cursor;
    bool right = is_bst(cursor->right, prev);

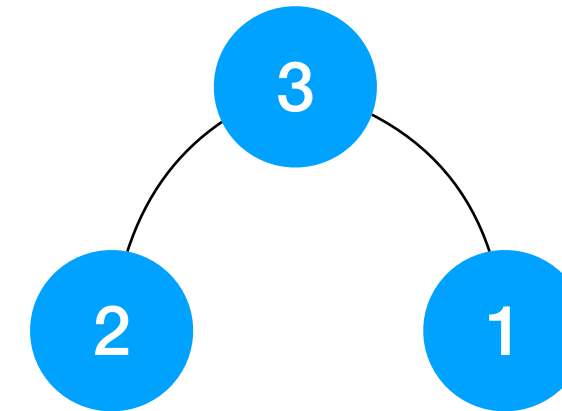
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	
NULL			

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    → bool right = is_bst(cursor->right, prev);

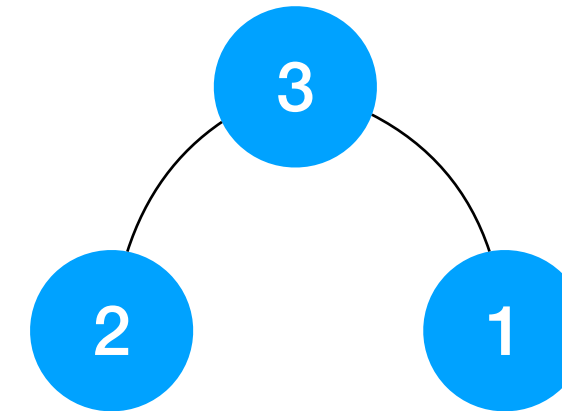
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	
NULL			

# Is this a BST?



```
→ bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

    return (left && right);
}
```

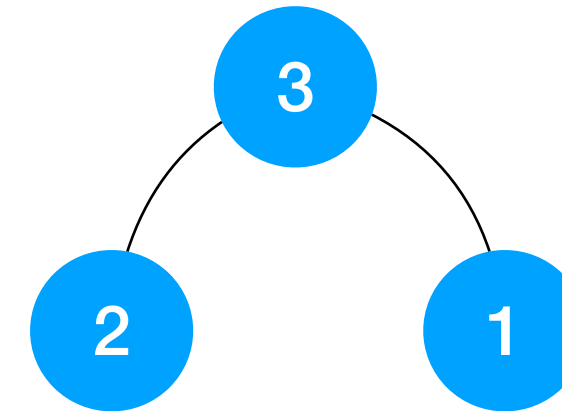
```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	
NULL			



# Is this a BST?



```
→ bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

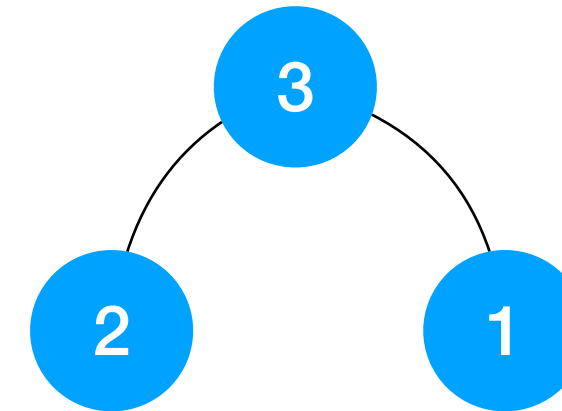
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	
NULL			
NULL	2		

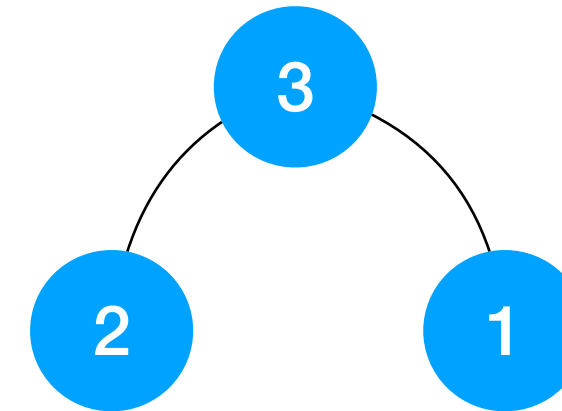
# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){  
→   if (cursor==NULL)  
       return true;  
  
   bool left = is_bst(cursor->left, prev);  
  
   if (prev!=NULL && cursor->data <= prev->data)  
       return false;  
  
   prev = cursor;  
   bool right = is_bst(cursor->right, prev);  
  
   return (left && right);  
}  
  
node *prev = NULL;  
  
if (is_bst(root, prev))  
    cout << "Tree is BST";  
else  
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	
NULL			
NULL	2		

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
    → return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

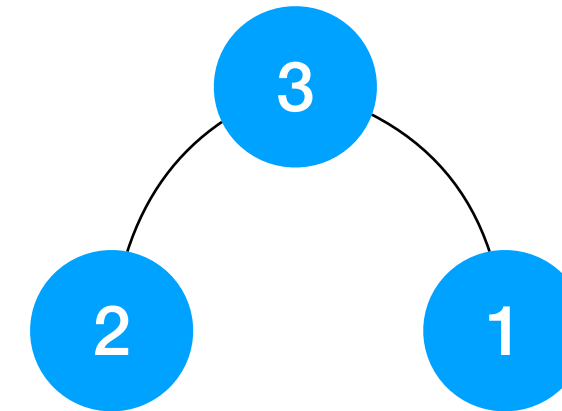
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	
NULL			
NULL	2		

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

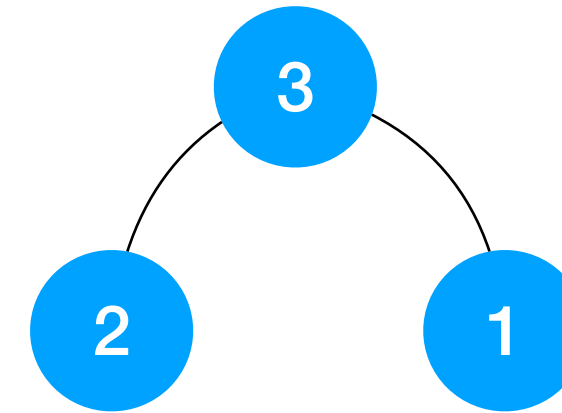
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	
NULL			
NULL	2		

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    → bool right = is_bst(cursor->right, prev);

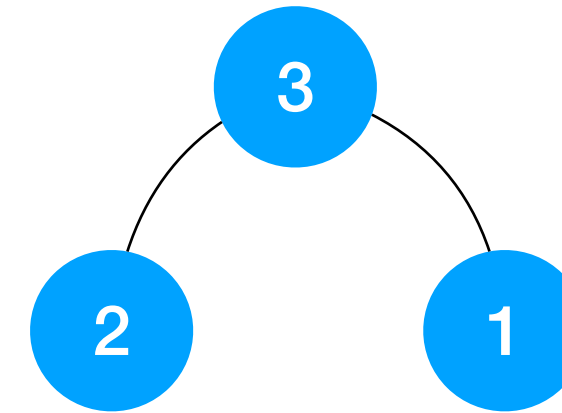
    return (left && right);
}

node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	true
NULL			
NULL	2		

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

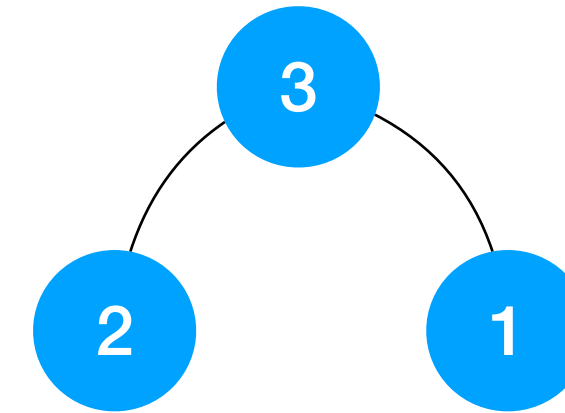
    prev = cursor;
    bool right = is_bst(cursor->right, prev);
    → return (left && right);
}

node *prev = NULL;

if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	true
NULL			
NULL	2		

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

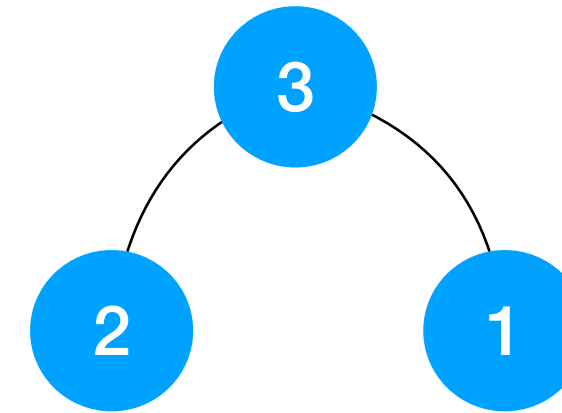
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	true
NULL			
NULL	2		

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;
    → bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

    return (left && right);
}
```

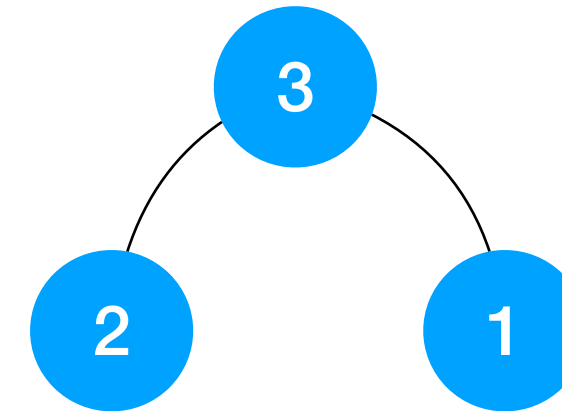
```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2		
2	2	true	true
NULL			
NULL	2		



# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;
    → bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

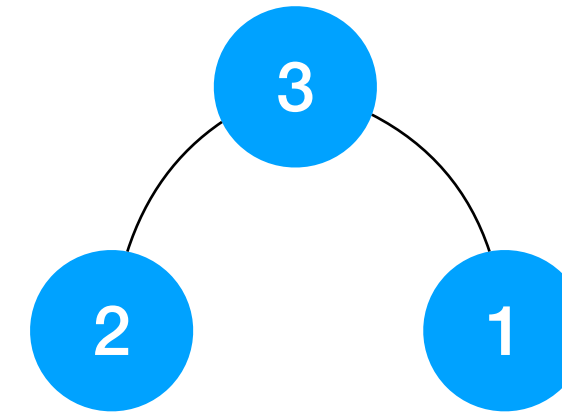
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2	true	
2	2	true	true
NULL			
NULL	2		

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    → if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

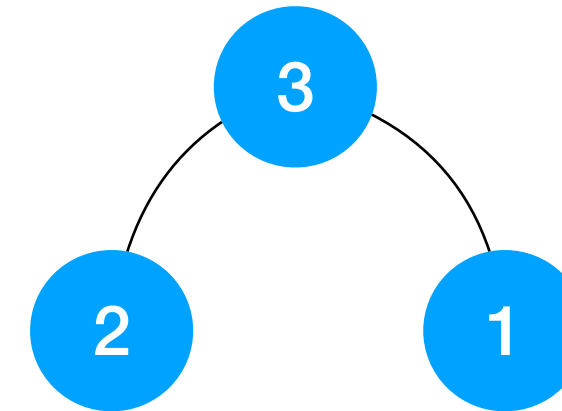
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2	true	
2	2	true	true
NULL			
NULL	2		

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    → prev = cursor;
    bool right = is_bst(cursor->right, prev);

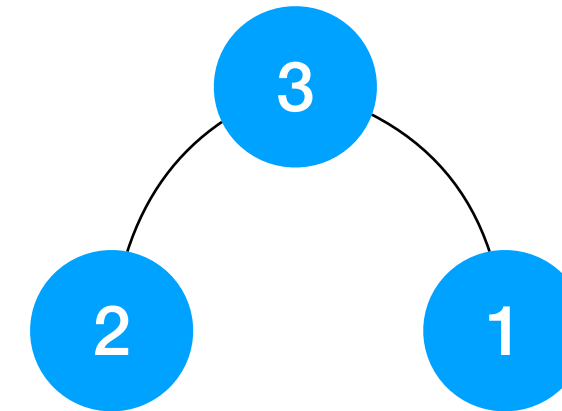
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	2	true	
2	2	true	true
NULL			
NULL	2		

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    → prev = cursor;
    bool right = is_bst(cursor->right, prev);

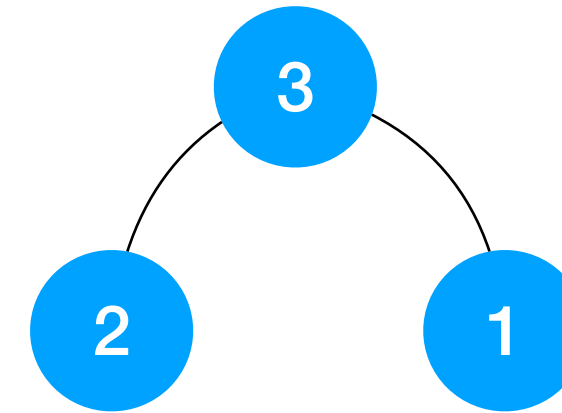
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	3	true	
2	3	true	true
NULL			
NULL	3		

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    → bool right = is_bst(cursor->right, prev);

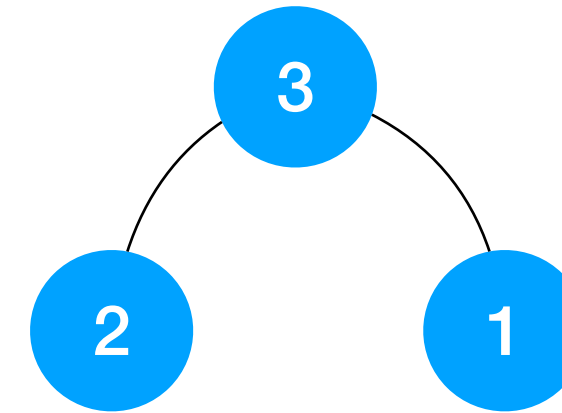
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	3	true	
2	3	true	true
NULL			
NULL	3		

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

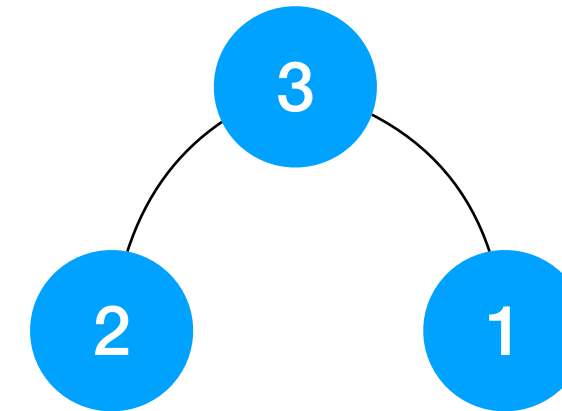
    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	3	true	
2	3	true	true
NULL			
NULL	3		

# Is this a BST?



```
bool is_bst(node *cursor, node *&prev){
    if (cursor==NULL)
        return true;

    bool left = is_bst(cursor->left, prev);

    if (prev!=NULL && cursor->data <= prev->data)
        return false;

    prev = cursor;
    bool right = is_bst(cursor->right, prev);

    return (left && right);
}
```

```
node *prev = NULL;
```

```
if (is_bst(root, prev))
    cout << "Tree is BST";
else
    cout << "Tree is not BST";
```

cursor	prev	left	right
3	3	true	
2	3	true	true
NULL			
NULL	3		

Keep going ... try it online: see [Gitlab](#)

# Other announcements

- Conflict exam policy (recap e-mails)



# Other announcements

- Conflict exam policy (recap e-mails)
- HKN review session (1230 - 1430 hrs, 12/12 in ECEB 1015)

# Other announcements

- Conflict exam policy (recap e-mails)
- HKN review session (1230 - 1430 hrs, 12/12 in ECEB 1015)
- Programming competition tomorrow at 7.00 pm in ECEB 1013.

# Other announcements

- Conflict exam policy (recap e-mails)
- HKN review session (1230 - 1430 hrs, 12/12 in ECEB 1015)
- Programming competition tomorrow at 7.00 pm in ECEB 1013.
- Additional study material is on the course website

# Other announcements

- Conflict exam policy (recap e-mails)
- HKN review session (1230 - 1430 hrs, 12/12 in ECEB 1015)
- Programming competition tomorrow at 7.00 pm in ECEB 1013.
- Additional study material is on the course website
  - Practice, practice, practice!

# Other announcements

- Conflict exam policy (recap e-mails)
- HKN review session (1230 - 1430 hrs, 12/12 in ECEB 1015)
- Programming competition tomorrow at 7.00 pm in ECEB 1013.
- Additional study material is on the course website
  - Practice, practice, practice!
- Exam format

# Last new topic/information

- Dynamic dispatch. Recall Bruno the cat and his lunch?

# Last new topic/information

- Dynamic dispatch. Recall Bruno the cat and his lunch?

```
#include <iostream>
using namespace std;

class Animal{
public:
    virtual void eat(){
        cout << "I'm eating generic food." << endl;
    }
};

class Cat : public Animal{
public:
    void eat(){
        cout << "I'm eating a mouse." << endl;
    }
};

void eat_lunch(Animal *a){
    a->eat();
}
```

# Last new topic/information

- Dynamic dispatch. Recall Bruno the cat and his lunch?

```
int main(){
    Animal *anim = new Animal();
    Cat *bruno = new Cat();
    anim->eat();
    bruno->eat();

    eat_lunch(anim);
    eat_lunch(bruno);
}
```

```
#include <iostream>
using namespace std;

class Animal{
public:
    virtual void eat(){
        cout << "I'm eating generic food." << endl;
    }
};

class Cat : public Animal{
public:
    void eat(){
        cout << "I'm eating a mouse." << endl;
    }
};

void eat_lunch(Animal *a){
    a->eat();
}
```



# Last new topic/information

- Dynamic dispatch. Recall Bruno the cat and his lunch?

```
int main(){
    Animal *anim = new Animal();
    Cat *bruno = new Cat();
    anim->eat();
    bruno->eat();

    eat_lunch(anim);
    eat_lunch(bruno);
}
```

How is this accomplished?

```
#include <iostream>
using namespace std;

class Animal{
public:
    virtual void eat(){
        cout << "I'm eating generic food." << endl;
    }
};

class Cat : public Animal{
public:
    void eat(){
        cout << "I'm eating a mouse." << endl;
    }
};

void eat_lunch(Animal *a){
    a->eat();
}
```

# Virtual functions

# Virtual functions

- Function to call determined at *runtime*.

# Virtual functions

- Function to call determined at *runtime*.
  - Called dynamic dispatch or linkage.

# Virtual functions

- Function to call determined at *runtime*.
  - Called dynamic dispatch or linkage.
  - Commonly accomplished using virtual function/method table.

# Virtual functions

- Function to call determined at *runtime*.
  - Called dynamic dispatch or linkage.
  - Commonly accomplished using virtual function/method table.
  - Key idea(s):

# Virtual functions

- Function to call determined at *runtime*.
  - Called dynamic dispatch or linkage.
  - Commonly accomplished using virtual function/method table.
- Key idea(s):
  - You can define pointers to functions also (see Gitlab).

# Virtual functions

- Function to call determined at *runtime*.
  - Called dynamic dispatch or linkage.
  - Commonly accomplished using virtual function/method table.
- Key idea(s):
  - You can define pointers to functions also (see Gitlab).
  - For each class with virtual functions or deriving from a class with virtual functions, a **vtable** is maintained.

Animal vTable

eat()

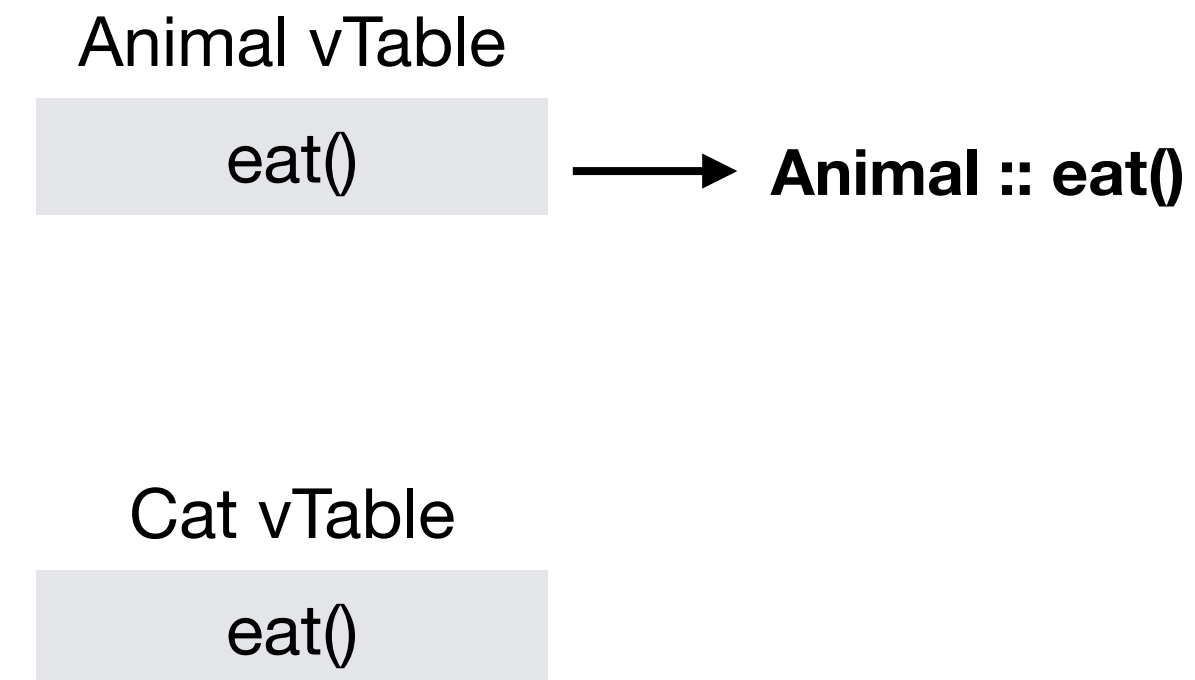
Cat vTable

eat()



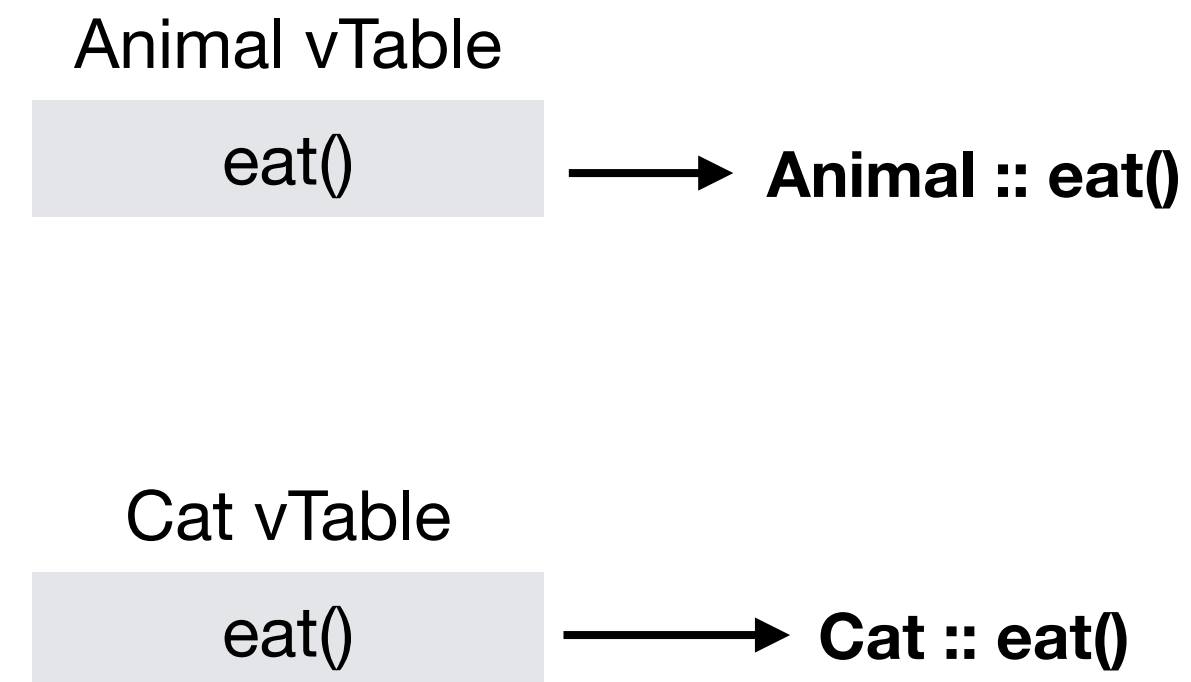
# Virtual functions

- Function to call determined at *runtime*.
  - Called dynamic dispatch or linkage.
  - Commonly accomplished using virtual function/method table.
- Key idea(s):
  - You can define pointers to functions also (see Gitlab).
  - For each class with virtual functions or deriving from a class with virtual functions, a **vtable** is maintained.



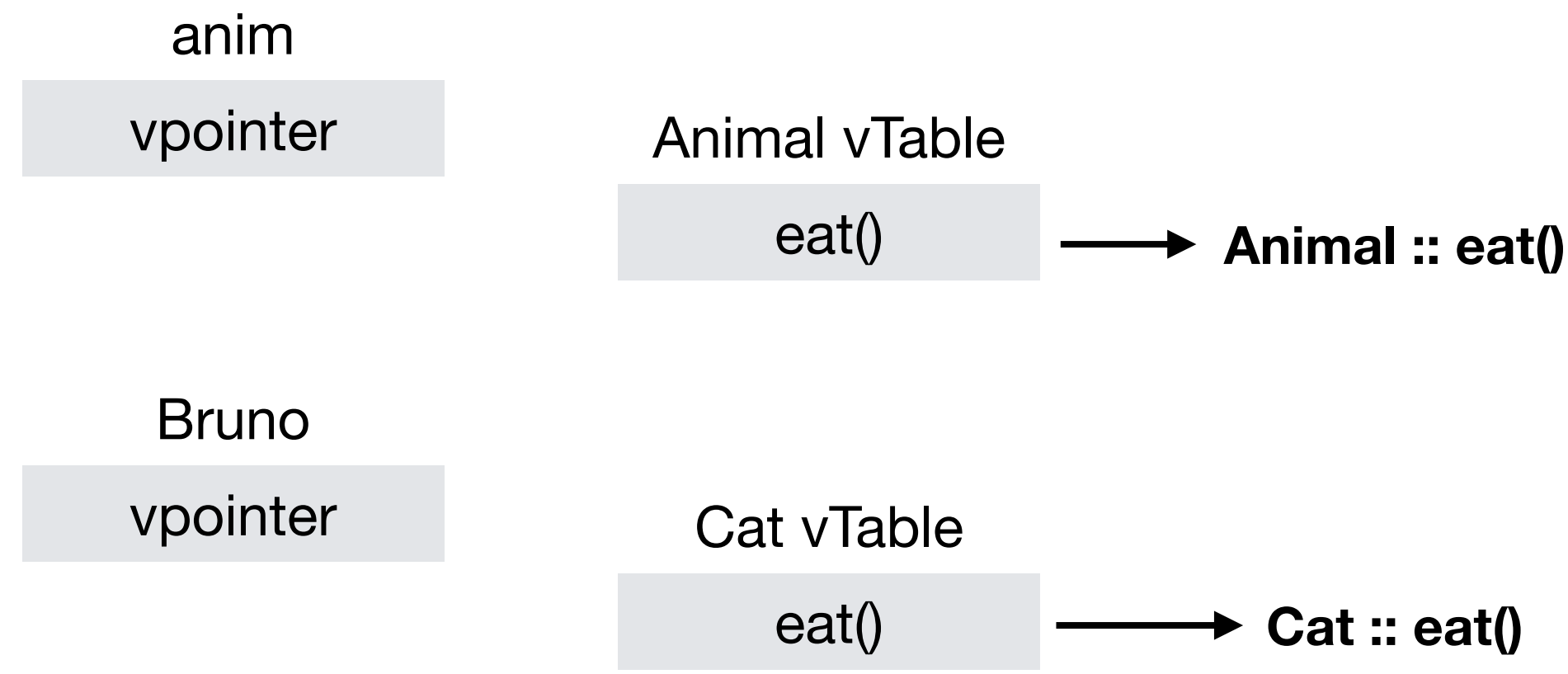
# Virtual functions

- Function to call determined at *runtime*.
  - Called dynamic dispatch or linkage.
  - Commonly accomplished using virtual function/method table.
- Key idea(s):
  - You can define pointers to functions also (see Gitlab).
  - For each class with virtual functions or deriving from a class with virtual functions, a **vtable** is maintained.



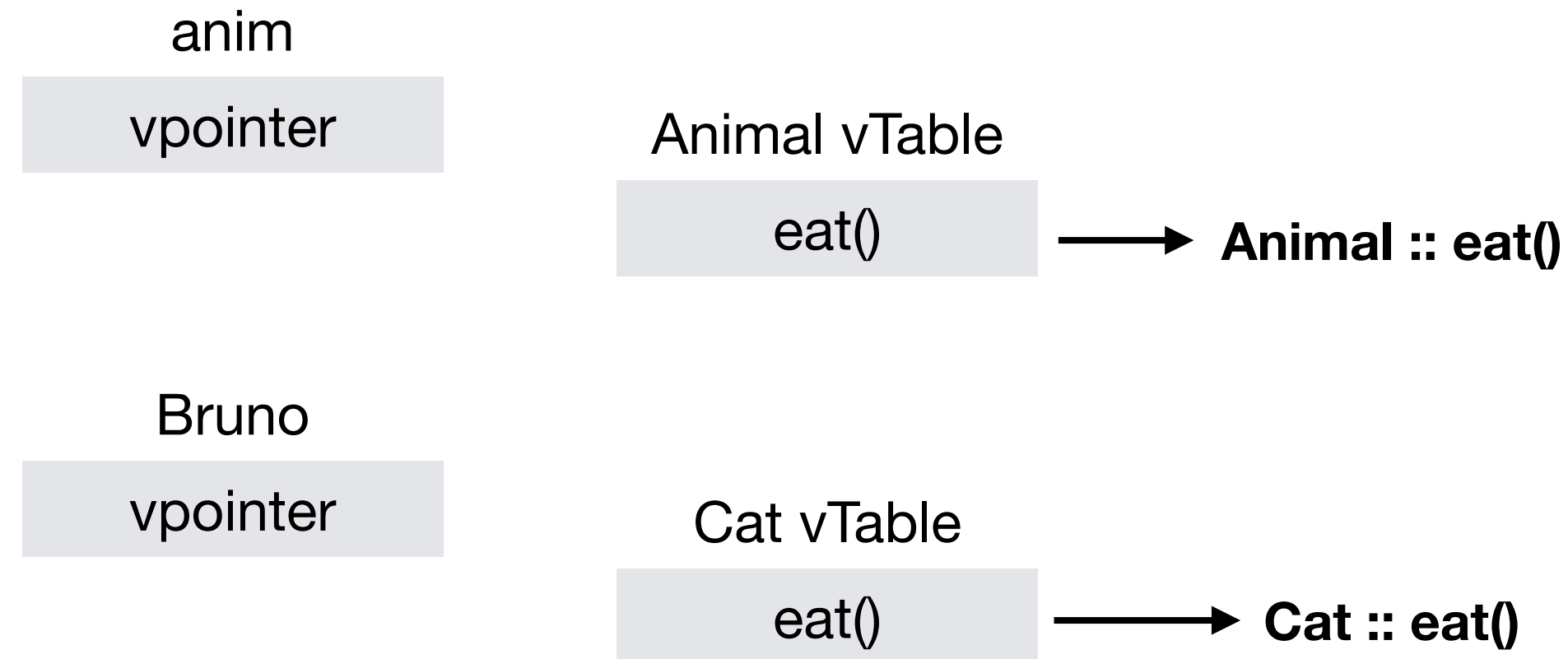
# Virtual functions

- Function to call determined at *runtime*.
  - Called dynamic dispatch or linkage.
  - Commonly accomplished using virtual function/method table.
- Key idea(s):
  - You can define pointers to functions also (see Gitlab).
  - For each class with virtual functions or deriving from a class with virtual functions, a **vtable** is maintained.
  - Compiler adds a pointer **vpointer** to this **vtable** to as data member to all objects.



# Virtual functions

- Function to call determined at *runtime*.
  - Called dynamic dispatch or linkage.
  - Commonly accomplished using virtual function/method table.
- Key idea(s):
  - You can define pointers to functions also (see Gitlab).
  - For each class with virtual functions or deriving from a class with virtual functions, a **vtable** is maintained.
  - Compiler adds a pointer **vpointer** to this **vtable** to as data member to all objects.

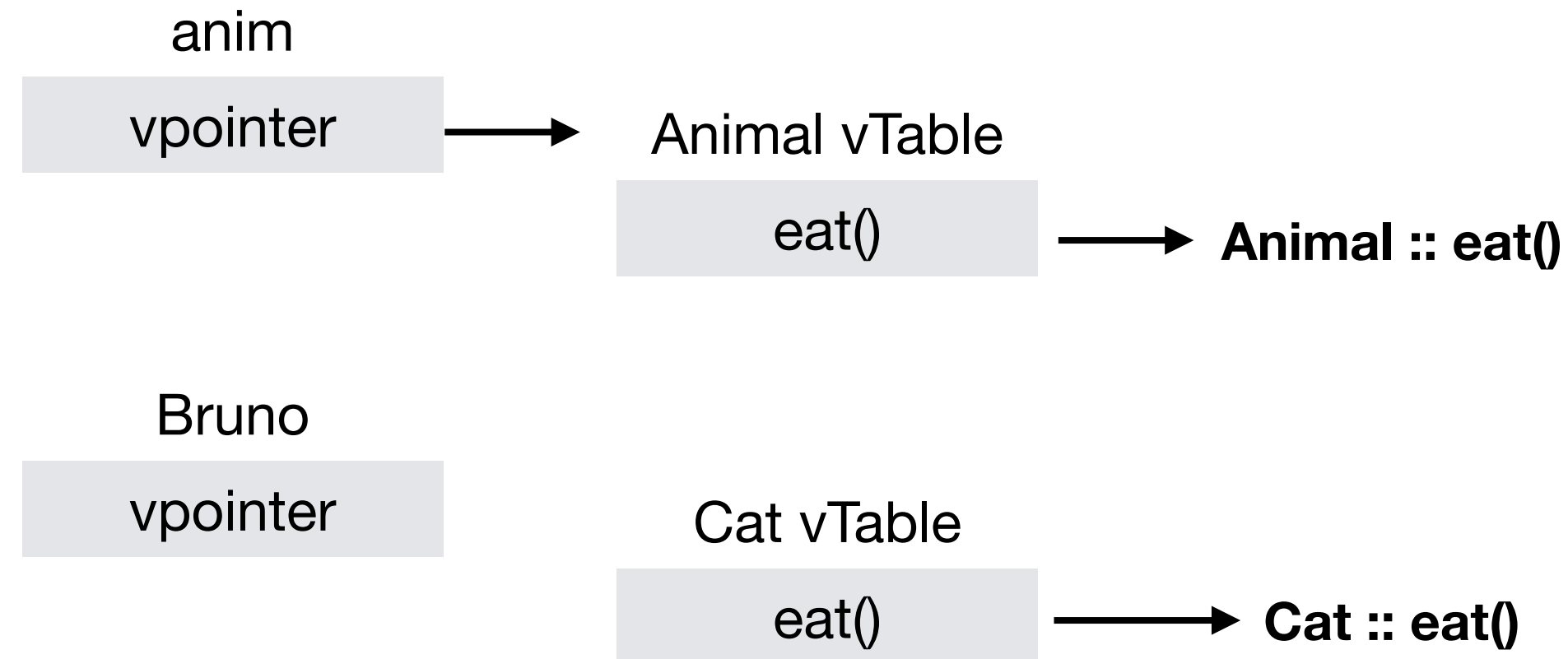


```
void eat_lunch(Animal *a){  
    a->eat();  
}
```

```
Cat *bruno = new Cat();  
eat_lunch(bruno);
```

# Virtual functions

- Function to call determined at *runtime*.
  - Called dynamic dispatch or linkage.
  - Commonly accomplished using virtual function/method table.
- Key idea(s):
  - You can define pointers to functions also (see Gitlab).
  - For each class with virtual functions or deriving from a class with virtual functions, a **vtable** is maintained.
  - Compiler adds a pointer **vpointer** to this **vtable** to as data member to all objects.

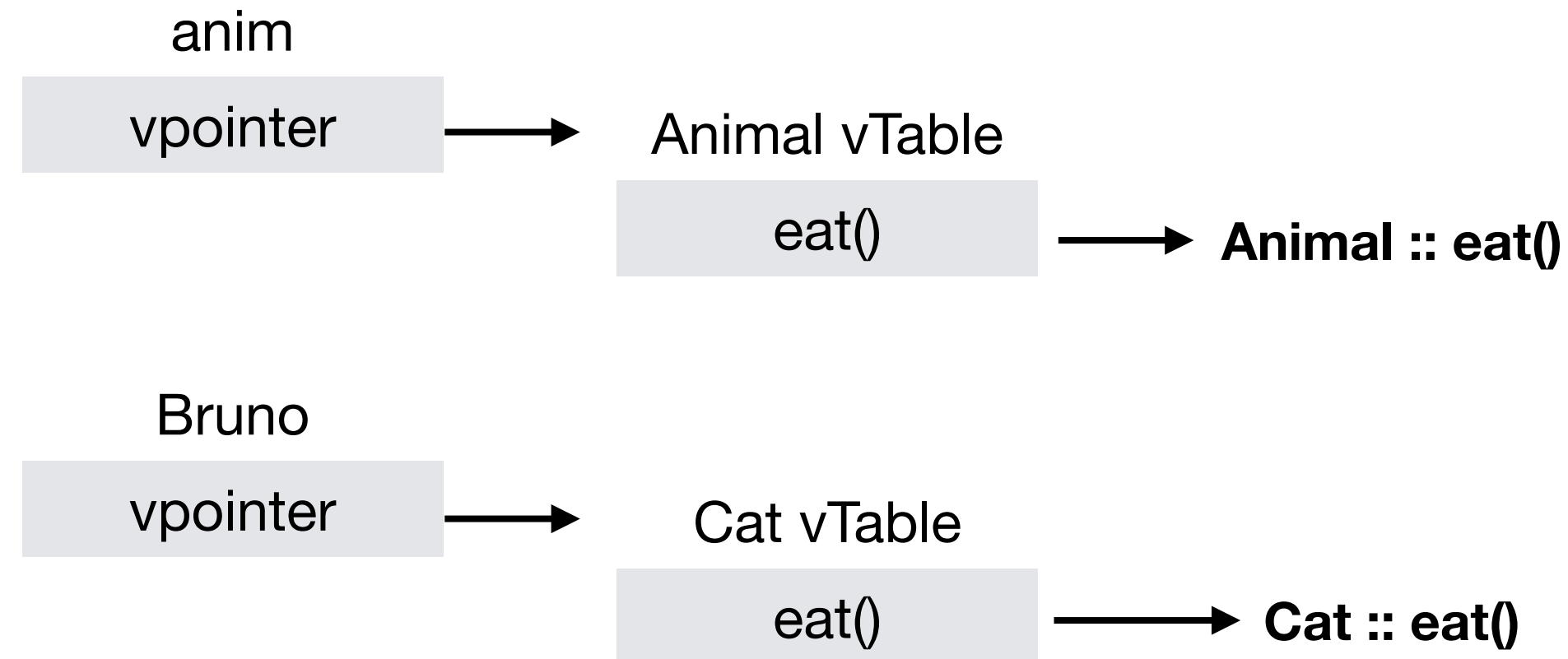


```
void eat_lunch(Animal *a){
    a->eat();
}
```

```
Cat *bruno = new Cat();
eat_lunch(bruno);
```

# Virtual functions

- Function to call determined at *runtime*.
  - Called dynamic dispatch or linkage.
  - Commonly accomplished using virtual function/method table.
- Key idea(s):
  - You can define pointers to functions also (see Gitlab).
  - For each class with virtual functions or deriving from a class with virtual functions, a **vtable** is maintained.
  - Compiler adds a pointer **vpointer** to this **vtable** to as data member to all objects.



```
void eat_lunch(Animal *a){  
    a->eat();  
}
```

```
Cat *bruno = new Cat();  
eat_lunch(bruno);
```

# Another example

```
// Base class
class Base {
public:
    virtual void function1(){
        cout << "Base function1()" << endl;
    }
    virtual void function2(){
        cout << "Base function2()" << endl;
    }
    virtual void function3(){
        cout << "Base function3()" << endl;
    }
};

// class derived from Base
class Derived1 : public Base {
public:
    // overriding function1()
    void function1(){
        cout << "Derived1 function1()" << endl;
    }
    // not overriding function2() and function3()
};
```

```
// class derived from Derived1
class Derived2 : public Derived1 {
public:
    // again overriding function2()
    void function2(){
        cout << "Derived2 function2()" << endl;
    }
    // not overriding function1() and function3()
};

// driver code
int main(){
    // defining base class pointers
    Base* ptr1 = new Base();
    Base* ptr2 = new Derived1();
    Base* ptr3 = new Derived2();

    return 0;
}
```

Source: <https://www.geeksforgeeks.org/vtable-and-vptr-in-cpp/>

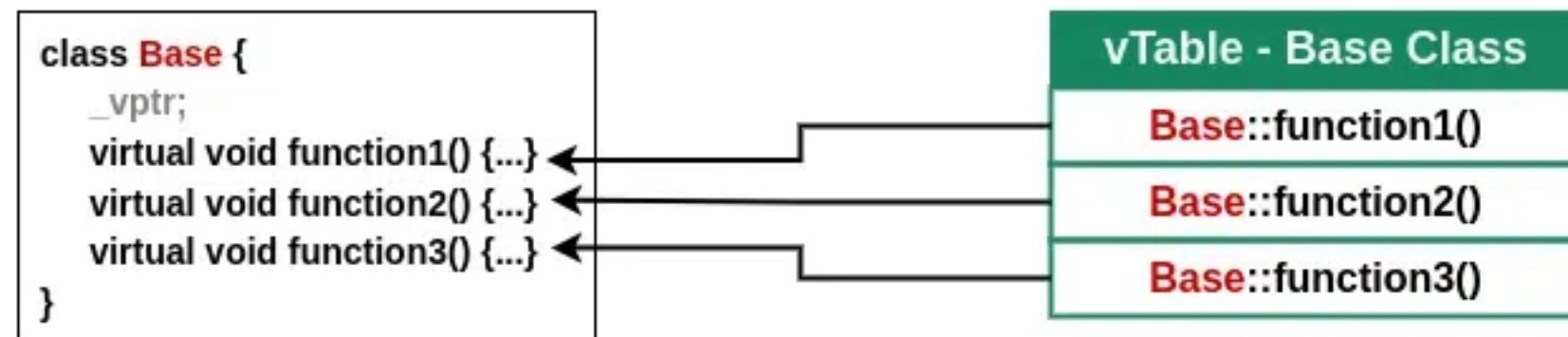
# Another example

```
// class derived from Base
class Derived1 : public Base {
public:
    // overriding function1()
    void function1(){
        cout << "Derived1 function1()" << endl;
    }
    // not overriding function2() and function3()
};
```

Source: <https://www.geeksforgeeks.org/vtable-and-vptr-in-cpp/>



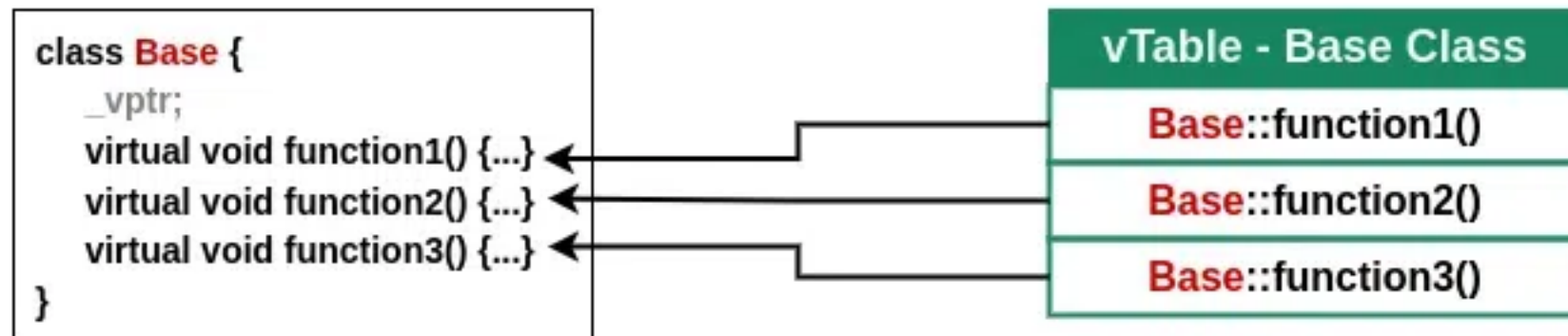
# Another example



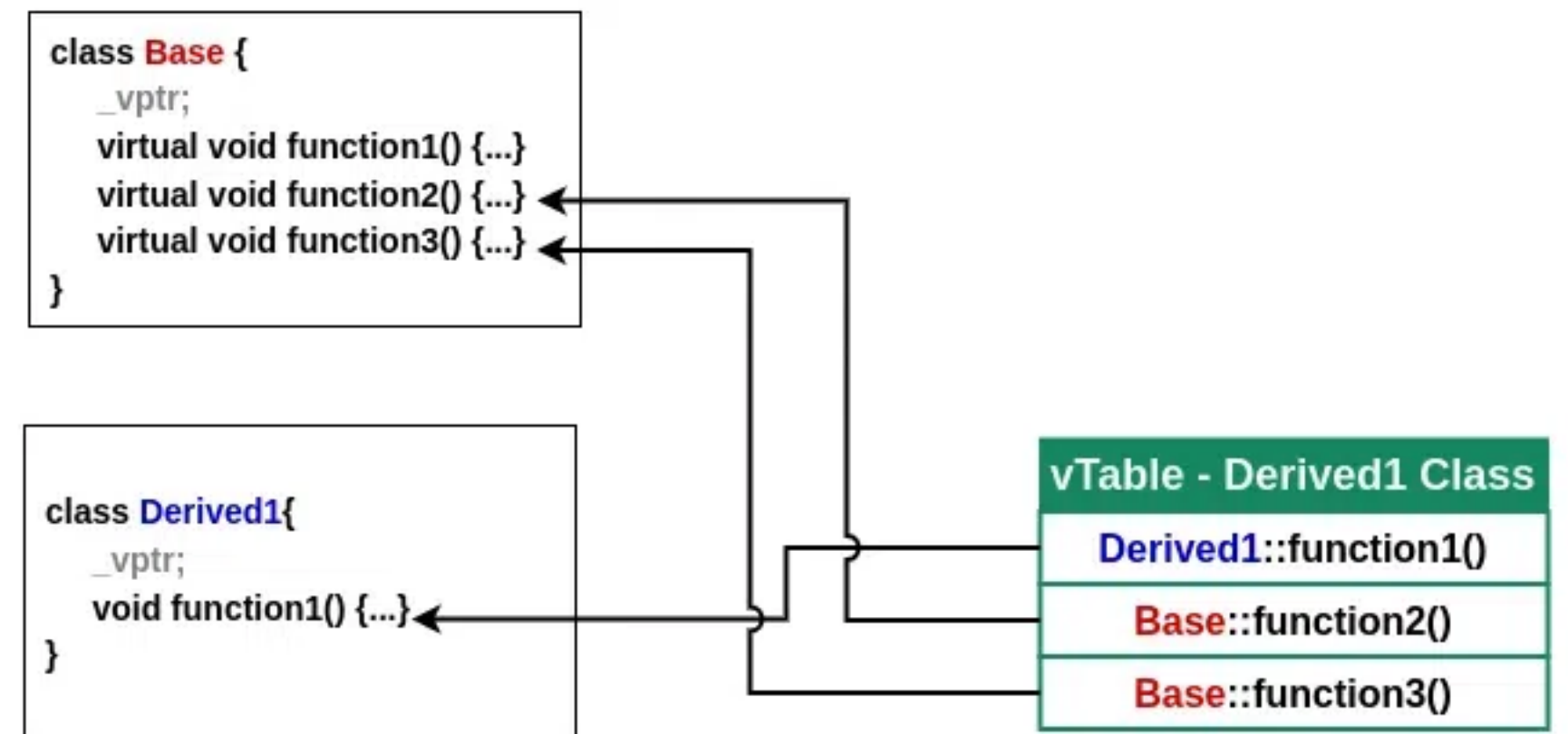
```
// class derived from Base  
class Derived1 : public Base {  
public:  
    // overriding function1()  
    void function1(){  
        cout << "Derived1 function1()" << endl;  
    }  
    // not overriding function2() and function3()  
};
```

Source: <https://www.geeksforgeeks.org/vtable-and-vptr-in-cpp/>

# Another example

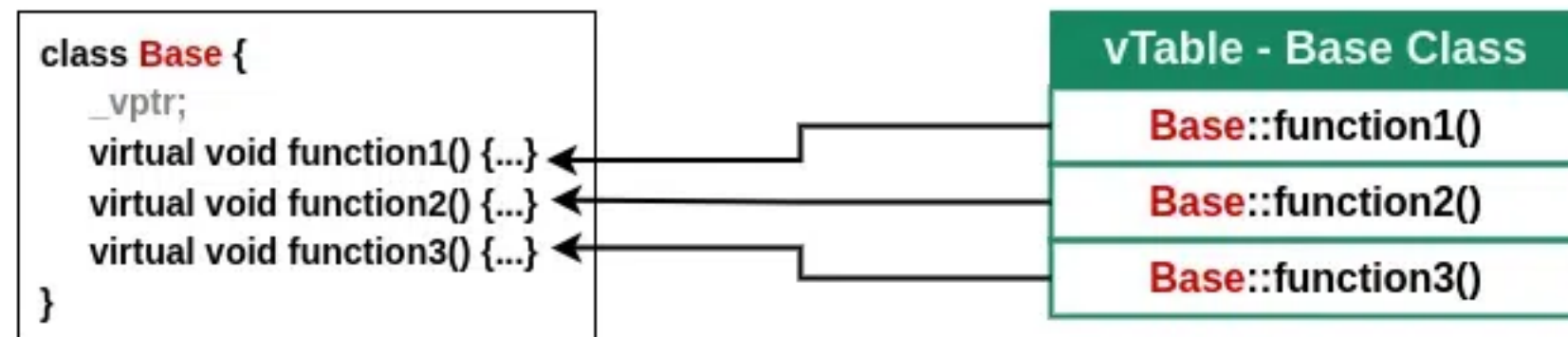


```
// class derived from Base  
class Derived1 : public Base {  
public:  
    // overriding function1()  
    void function1(){  
        cout << "Derived1 function1()" << endl;  
    }  
    // not overriding function2() and function3()  
};
```



Source: <https://www.geeksforgeeks.org/vtable-and-vptr-in-cpp/>

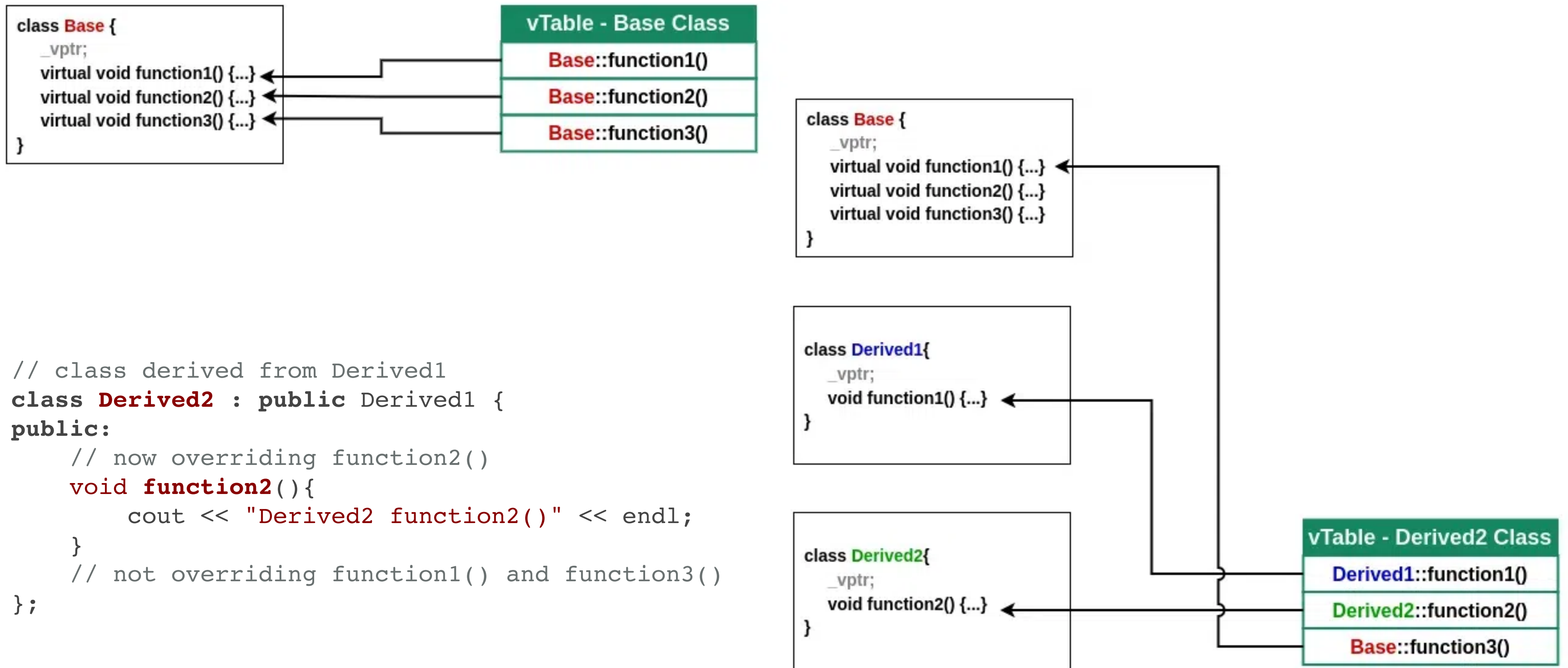
# Another example



```
// class derived from Derived1
class Derived2 : public Derived1 {
public:
    // now overriding function2()
    void function2(){
        cout << "Derived2 function2()" << endl;
    }
    // not overriding function1() and function3()
};
```

Source: <https://www.geeksforgeeks.org/vtable-and-vptr-in-cpp/>

# Another example



Source: <https://www.geeksforgeeks.org/vtable-and-vptr-in-cpp/>

# Topics to review ...

## Part 1: LC-3

- Assembly language programming & process
- Memory-mapped I/O: input from keyboard, output to monitor
- TRAPs & Subroutines, Interrupts & Exceptions
- Stacks

## Part 2: C

- Built-in data types, operators, scope
- Functions & run-time stack
- Pointers & arrays
- Recursion: searching, sorting, backtracking

- I/O: streams and buffers, read from / write to file
- User-defined data types: enum, struct, union
- Dynamic memory allocation
- Linked data structures: linked list (stack, queue) & trees

## Part 3: C++

- Class (encapsulation, inheritance, abstraction)
- Pass by value /(const) reference / address
- Virtual function, operator overload, template (polymorphism)

# Part 1 - LC3

- Address space: 2<sup>16</sup> locations, addressability: 16 bits
- General-purpose registers: R0, R1, ... R7
- Special-purpose register: PC, IR
- Input from keyboard: KBDR/KBSR
- Output to monitor: DDR/DSR
- Operate instructions: ADD, AND, NOT
- Data movement instructions: LD, LDI, LDR, LEA, ST, STR, STI
- Control instructions: BR, JSR/JSRR, JMP, RET, TRAP, RTI
- Condition codes: N (negative), Z (zero), P (positive)
- TRAPs: In, GETC, OUT, PUTS (uses R0; R7 is modified after call)
- Subroutines: callee-save vs. caller-save, nested subroutine needs to save R7
- Interrupts: external event, supervisor vs. user stack, RTI instruction
- Exceptions: internal event for handling errors
- Stack: FILO, overflow, underflow, R6 – stack ptr, R5 – frame ptr

# Part 2 - C language

- Scope: local vs. global variables (determined by location of declaration)
- Storage class: static (retains value, global data area) vs. automatic (stack)
- Control structures: conditionals (if, if-else, switch); loops (for, while, do-while)
- Functions & run-time stack (C to LC-3)
- Pointer: address of a variable in memory
- Array: a list of values arranged sequentially in memory
- Pass by value vs. pass by reference (pointer)
- Pointer Array Duality (`int array[10] = {1,2,3,4,5,6,7,8,9}; int *ptr = array;`)
- Recursion: base case(s) & recursive case(s)
- File I/O: `fopen`, `fclose`, `fscanf`, `fprintf`
- Linked lists & trees (pointer, struct, dynamic memory allocation)

# Part 3 - C++ language

- Class vs. struct: 4 features of OOP (polymorphism, inheritance, encapsulation, abstraction)
- Dynamic memory allocation: new & delete
- Basic I/O: std, cin, cout
- Pass by value vs. pass by address vs. pass by (const) reference
- Operator and function overloading
- Base class & derived Class: access identifier (public, protected, private)
- Virtual function & virtual function table: static vs. dynamic binding
- Function and class templates: separate type with container
- Big three: copy constructor (deep vs. shallow copy), destructor, copy assignment operator
- Implicit 'this' pointer: a pointer to the invoking object
- Vectors: dynamic arrays, elements are stored in consecutive locations
- Lists: doubly linked lists, elements are allocated individually
- Iterators: the mechanism used to minimize an algorithm's dependency on the data structure on which it operates



# Practice material

Let us do some posted practice material/questions ...