# TDDE18 & 726G77

## Classes & Pointers

Christoffer Holm

Department of Computer and information science

LINKÖPING UNIVERSITY

1 Classes
2 List lab
3 Special Member Functions

# Classes

What is a class?

```
struct Date
{
  int day;
  int month;
  int year;
};


bool operator<(Date d1, Date d2)
{
  // ...
}

Date next_date(Date d)
{
  // ...
}
```

# Classes

What is a class?

```
struct Date
{
  int day;
  int month;
  int year;
};


bool operator<(Date d1, Date d2)
{
  // ...
}

Date next_date(Date d)
{
  // ...
}
```

```
class Date
{
public:
  int day;
  int month;
  int year;

  bool operator<(Date d)
  {
    // ...
  }

  Date next_date()
  {
    // ...
  }
};
```

# Classes

How does it work?

```
Date today {27, 9, 2019};
Date tomorrow {next_date(today)};

if (today < tomorrow)
{
  // ...
}
```

# Classes

How does it work?

```
Date today {27, 9, 2019};
Date tomorrow {next_date(today)};

if (today < tomorrow)
{
  // ...
}
```

```
Date today {27, 9, 2019};
Date tomorrow {today.next_date()};

if (today < tomorrow)
{
  // ...
}
```

# Classes

How does it work?

```
Date today {27, 9, 2019};
Date tomorrow {next_date(today)};

if (operator<(today, tomorrow))
{
  // ...
}
```

```
Date today {27, 9, 2019};
Date tomorrow {today.next_date()};

if (today.operator<(tomorrow))
{
  // ...
}
```

# Classes

## this

```
struct Date
{
  // ...
};

void increase_year(Date& date)
{
  ++(date.year);
}
// ...
```

```
class Date
{
  // ...

  void increase_year()
  {
    ++(this->year);
  }
  // ...
};
```

# Classes

## this

```
struct Date
{
  // ...
};

void increase_year(Date& date)
{
  ++(date.year);
}
// ...
```

```
class Date
{
  // ...

  void increase_year()
  {
    ++year;
  }
  // ...
};
```

# Classes

When is `this` mandatory?

```
class Date
{
  // ...
  void set_year(int year)
  {
    year = year;
  }
};
```

# Classes

When is `this` mandatory?

```
class Date
{
  // ...
  void set_year(int year)
  {
    this->year = year;
  }
};
```

# Classes

When is `this` mandatory?

```
class Date
{
  // ...
  void set_year(int y)
  {
    year = y;
  }
};
```

# Classes

Why though?

```
Date today {27, 9, 2019};

today.day = 48; // should not be allowed
```

# Classes

## `private` & `public`

```
class Date
{
public:
  bool operator<(Date const& d) const
  {
    // ...
  }

  Date next_date()
  {
    // ...
  }

private:
  int day;
  int month;
  int year;
};
```

# Classes

## `private` & `public`

```
class Date
{
public:
  bool operator<(Date const& d) const
  {
    // ...
  }

  Date next_date()
  {
    // ...
  }

private:
  int day;
  int month;
  int year;
};
```

```
int main()
{

  Date today {27, 9, 2019};
  // not allowed
  today.day = 48;
}
```

# Classes

## `private` & `public`

```
class Date
{
public:
  bool operator<(Date const& d) const
  {
    // ...
  }

  Date next_date()
  {
    // ...
  }

private:
  int day;
  int month;
  int year;
};
```

```
int main()
{
  // will not work
  Date today {27, 9, 2019};
  // not allowed
  today.day = 48;
}
```

# Classes

## Constructors

```
class Date
{
public:
  Date(int d, int m, int y)
    // member initialization list
    : day{d}, month{m}, year{y}
  {
  }

  // ...

private:
  int day;
  int month;
  int year;
};
```

# Classes

## Constructors

```
class Date
{
public:
  Date(int d, int m, int y)
    // member initialization list
    : day{d}, month{m}, year{y}
  {
  }

  // ...

private:
  int day;
  int month;
  int year;
};
```

```
int main()
{
  // works!
  Date today {27, 9, 2019};
  // not allowed
  today.day = 48;
}
```

# Classes

## Declaration & Definition

```
class Date;                     // class declaration
class Date                      // class definition
{
  // ...
  Date(int d, int m, int y); // declare a construtor
  void increase_year();      // declare a member function
  // ...
private:                     // data members
  int day;
  int month;
  int year;
};

Date::Date(int d, int m, int y) // define a construtor
  : day{d}, month{m}, year{y}   // member initialization list
{ }

void Date::increase_year()      // define a member function
{
  ++year;
}
```

# Classes

`const` member functions

```
class Date
{
  // ...
  int get_day()
  {
    day = 7; // allowed
    return day;
  }
};
```

# Classes

`const` member functions

```
class Date
{
  // ...
  int get_day() const
  {
    day = 7; // NOT allowed
    return day;
  }
};
```

# Classes

## `const` member functions

```
class Date
{
  // ...
  int get_day()
  {
    return day;
  }
  // ...
};
```

```
Date d1{27, 9, 2019};
cout << d1.get_day() << endl;

Date const d2{28, 9, 2019};

// doesn't work
cout << d2.get_day() << endl;
```

# Classes

## `const` member functions

```
class Date
{
  // ...
  int get_day() const
  {
    return day;
  }
  // ...
};
```

```
Date d1{27, 9, 2019};
cout << d1.get_day() << endl;

Date const d2{28, 9, 2019};

// works!
cout << d2.get_day() << endl;
```

# Classes

## Inner class

```
class Outer
{
public:

  void fun();



  class Inner
  {
  public:

    void fun();

  };

};
```

```
void Outer::fun()
{
  // ...
}

void Outer::Inner::fun()
{
  // ...
}
```

```
Outer o{};
o.fun();

Outer::Inner i{}; // works!
i.fun();
```

# Classes

## Inner class

```
class Outer
{
public:

  void fun();

private:

  class Inner
  {
  public:

    void fun();

  };

};
```

```
void Outer::fun()
{
  // ...
}

void Outer::Inner::fun()
{
  // ...
}
```

```
Outer o{};
o.fun();

Outer::Inner i{}; // doesn't work
i.fun();
```

# Classes

## friend

```
bool same_month(Date d1, Date d2);
class Date
{
  // ...

private:

  int day;
  int month;
  int year;

  friend bool same_month(Date d1, Date d2);
};

bool same_month(Date d1, Date d2)
{
  return d1.year == d2.year && d1.month == d2.month;
}
```

LINKÖPING UNIVERSITY

# List lab

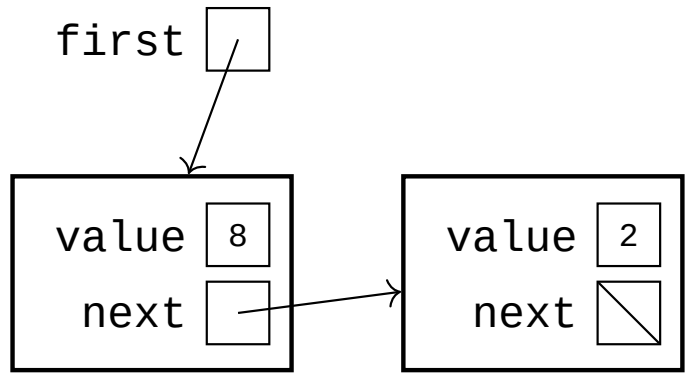
Node

```
struct Node
{
  int value;
  Node* next;
};

Node n2 {2, nullptr};
Node n1 {8, &n2};

Node* first {&n1};
```

# List lab

Node

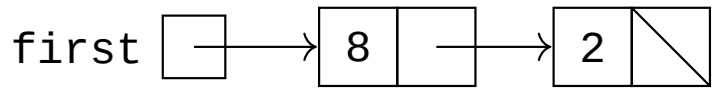
```
struct Node
{
  int value;
  Node* next;
};

Node n2 {2, nullptr};
Node n1 {8, &n2};

Node* first {&n1};
```



# List lab

Node

```
struct Node
{
  int value;
  Node* next;
};

Node n2 {2, nullptr};
Node n1 {8, &n2};

Node* first {&n1};
```



# List lab

Accessing data in `Node`

```
Node n2 {2, nullptr};
Node n1 {8, &n2};

Node* first {&n1};

cout << (*first).value << endl;
```

# List lab

Accessing data in `Node`

```
Node n2 {2, nullptr};
Node n1 {8, &n2};

Node* first {&n1};

cout << first->value << endl;
```

# List lab

`List`

```
class List
{
public:

  // ...

private:
  Node* first{};
};
```

# List lab

## insert

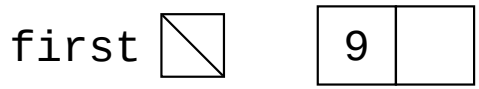
```
class List
{
public:

  void remove();
  void insert(int value)
  {
    Node* tmp{new Node{value}};
    tmp->next = first;
    first = tmp;
  }

private:
  Node* first{};
};
```

first 

# List lab

## insert

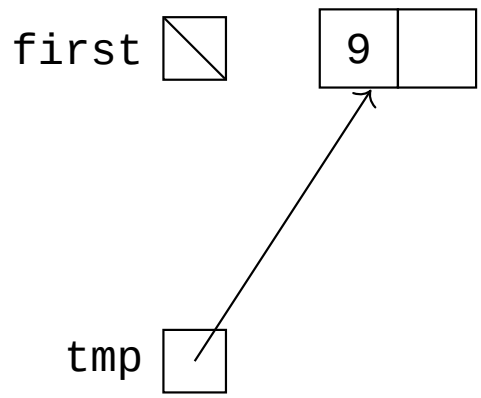
```
class List
{
public:

  void remove();
  void insert(int value)
  {
    Node* tmp{new Node{value}};
    tmp->next = first;
    first = tmp;
  }

private:
  Node* first{};
};
```



# List lab

## insert

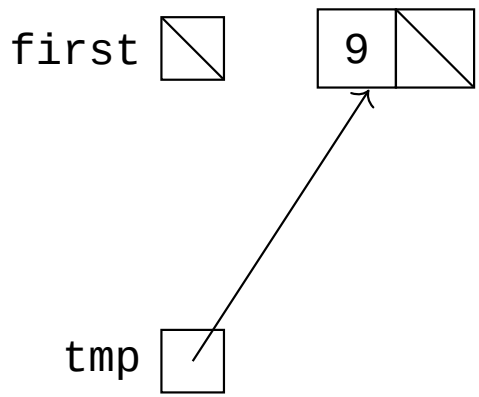
```
class List
{
public:

  void remove();
  void insert(int value)
  {
    Node* tmp{new Node{value}};
    tmp->next = first;
    first = tmp;
  }

private:
  Node* first{};
};
```



# List lab

## insert

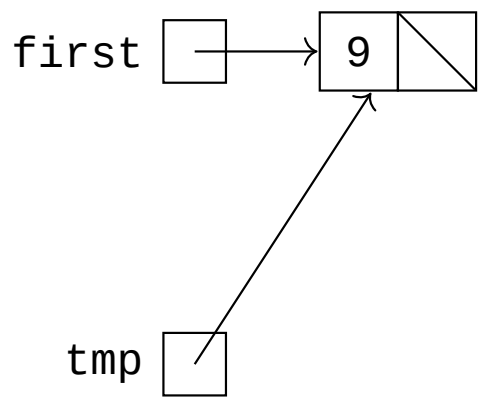
```
class List
{
public:

  void remove();
  void insert(int value)
  {
    Node* tmp{new Node{value}};
    tmp->next = first;
    first = tmp;
  }

private:
  Node* first{};
};
```



# List lab

## insert

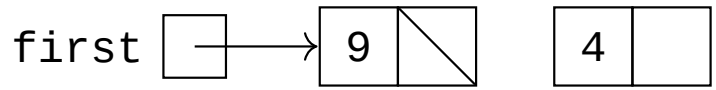
```
class List
{
public:

  void remove();
  void insert(int value)
  {
    Node* tmp{new Node{value}};
    tmp->next = first;
    first = tmp;
  }

private:
  Node* first{};
};
```



# List lab

## insert

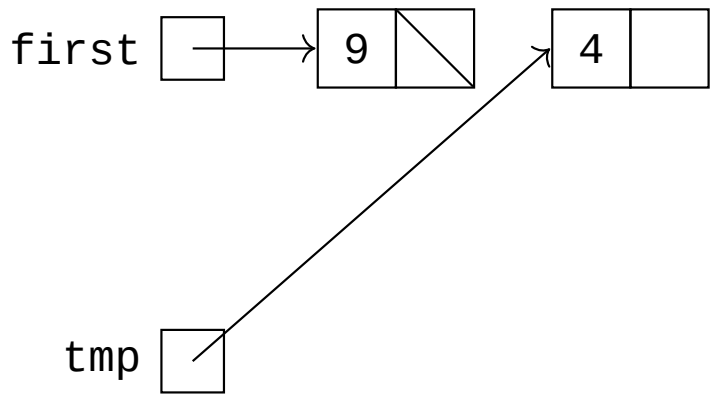
```
class List
{
public:

  void remove();
  void insert(int value)
  {
    Node* tmp{new Node{value}};
    tmp->next = first;
    first = tmp;
  }

private:
  Node* first{};
};
```



# List lab

## insert

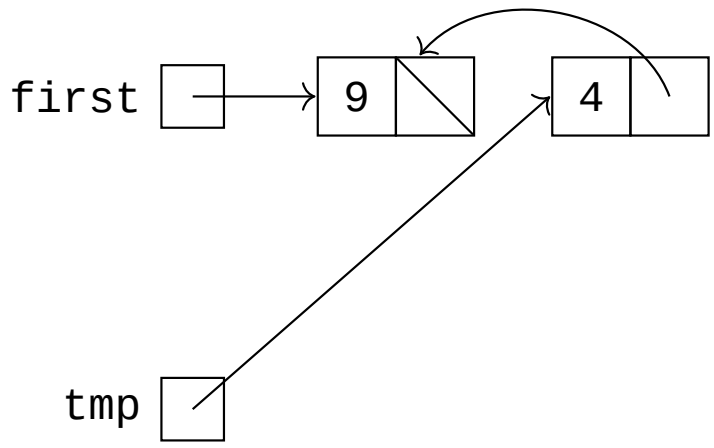
```
class List
{
public:

  void remove();
  void insert(int value)
  {
    Node* tmp{new Node{value}};
    tmp->next = first;
    first = tmp;
  }

private:
  Node* first{};
};
```



# List lab

## insert

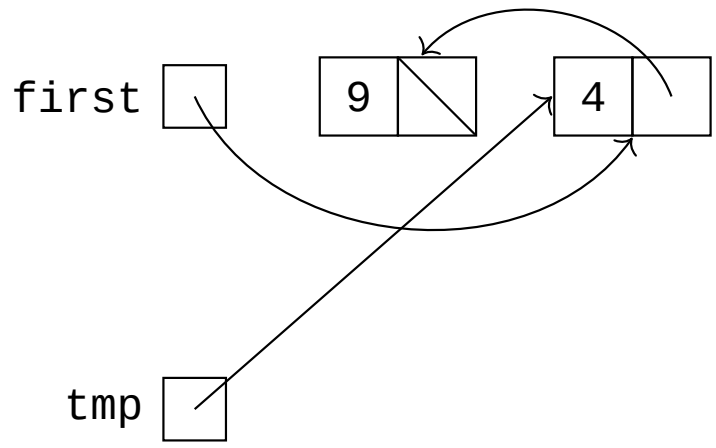
```
class List
{
public:

  void remove();
  void insert(int value)
  {
    Node* tmp{new Node{value}};
    tmp->next = first;
    first = tmp;
  }

private:
  Node* first{};
};
```



# List lab

## insert

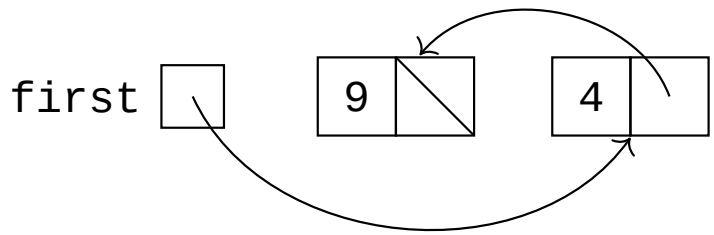
```
class List
{
public:

  void remove();
  void insert(int value)
  {
    Node* tmp{new Node{value}};
    tmp->next = first;
    first = tmp;
  }

private:
  Node* first{};
};
```



# List lab

## insert

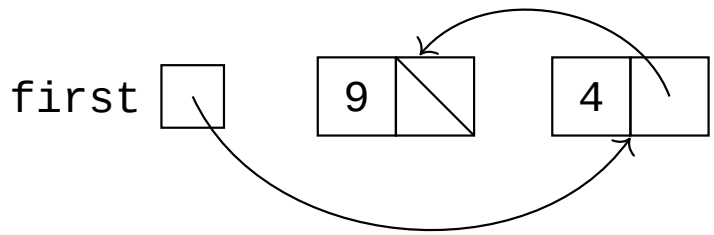
```
class List
{
public:

  void remove();
  void insert(int value)
  {
    Node* tmp{new Node{value}};
    tmp->next = first;
    first = tmp;
  }

private:
  Node* first{};
};
```



# List lab

A problem!

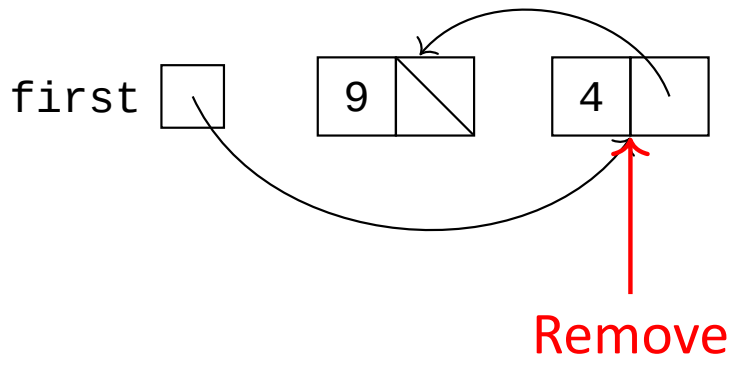
```
class List
{
public:

  void remove()
  {
    first = first->next;
  }
  void insert(int value);

private:
  Node* first{};
};
```



# List lab

A problem!

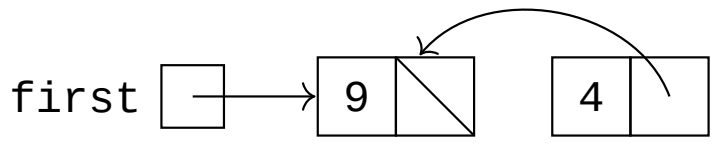
```
class List
{
public:

  void remove()
  {
    first = first->next;
  }
  void insert(int value);

private:
  Node* first{};
};
```



# List lab

A problem!

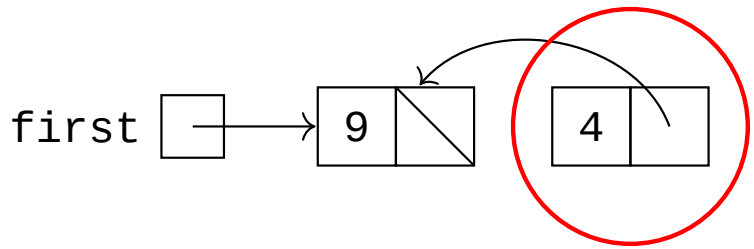
```
class List
{
public:

  void remove()
  {
    first = first->next;
  }
  void insert(int value);

private:
  Node* first{};
};
```



# List lab

A problem!

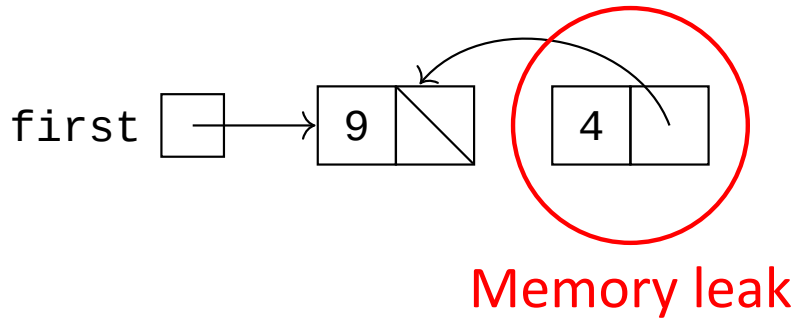
```
class List
{
public:

  void remove()
  {
    first = first->next;
  }
  void insert(int value);

private:
  Node* first{};
};
```



# List lab

A problem!

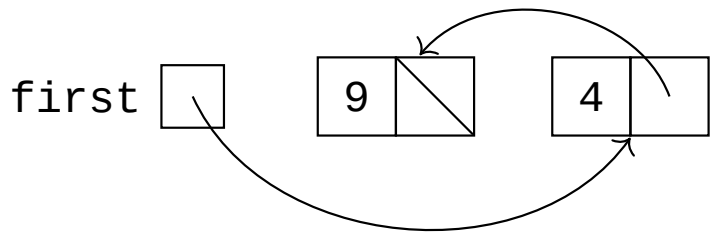
```
class List
{
public:

  void remove()
  {
    first = first->next;
  }
  void insert(int value);

private:
  Node* first{};
};
```



# List lab

Let’s try again!

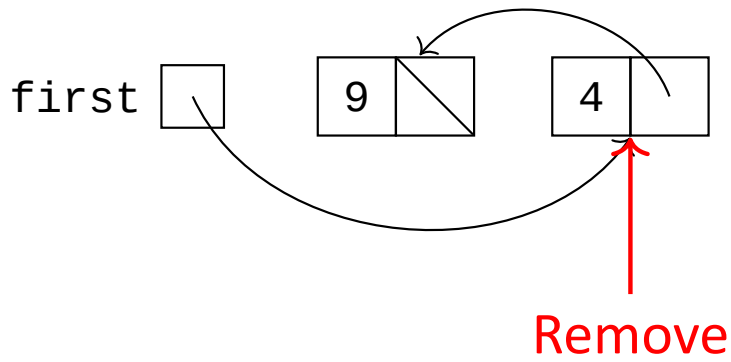
```
class List
{
public:

  void remove()
  {
    Node* tmp = first;
    first = first->next;
    delete tmp;
  }
  void insert(int value);

private:
  Node* first{};
};
```



# List lab

Let’s try again!

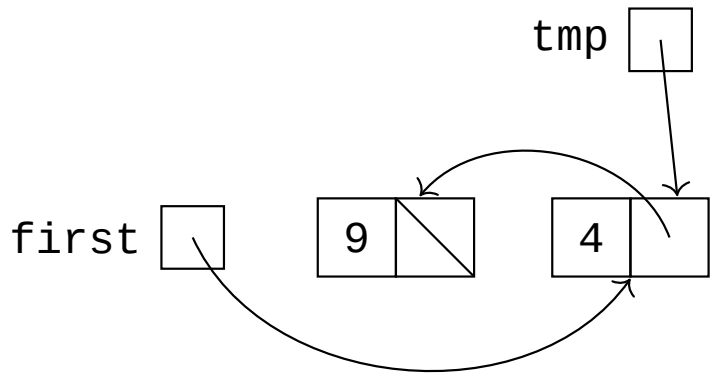
```
class List
{
public:

  void remove()
  {
    Node* tmp = first;
    first = first->next;
    delete tmp;
  }
  void insert(int value);

private:
  Node* first{};
};
```



# List lab

Let’s try again!

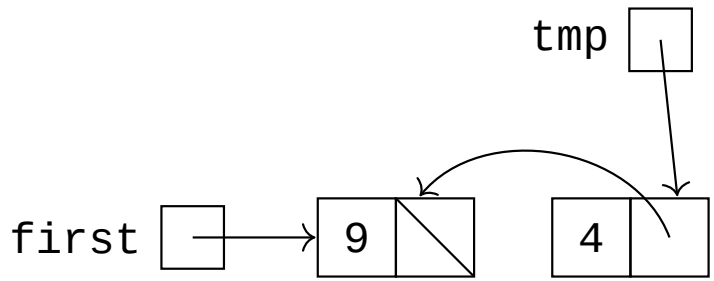
```
class List
{
public:

  void remove()
  {
    Node* tmp = first;
    first = first->next;
    delete tmp;
  }
  void insert(int value);

private:
  Node* first{};
};
```



# List lab

Let’s try again!

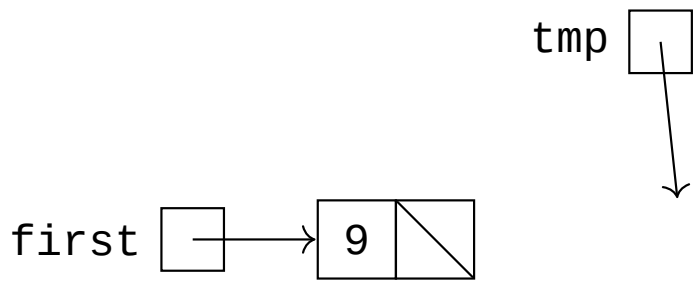
```
class List
{
public:

  void remove()
  {
    Node* tmp = first;
    first = first->next;
    delete tmp;
  }
  void insert(int value);

private:
  Node* first{};
};
```



# List lab

Let’s try again!

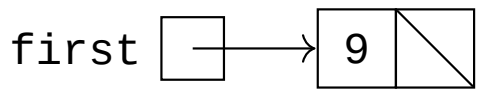
```
class List
{
public:

  void remove()
  {
    Node* tmp = first;
    first = first->next;
    delete tmp;
  }
  void insert(int value);

private:
  Node* first{};
};
```



# List lab

Let’s try again!

```
class List
{
public:

  void remove()
  {
    Node* tmp = first;
    first = first->next;
    delete tmp;
  }
  void insert(int value);

private:
  Node* first{};
};
```



1 Classes
2 List lab
3 Special Member Functions

# Special Member Functions

Destructor

```
class List
{
public:

  List() // constructor
    : first{nullptr}
  {
  }

  ~List() // destructor
  {
    // go through each node in our list and call delete on them
  }

  void remove();
  void insert(int value);

private:
  Node* first{};
};
```

# Special Member Functions

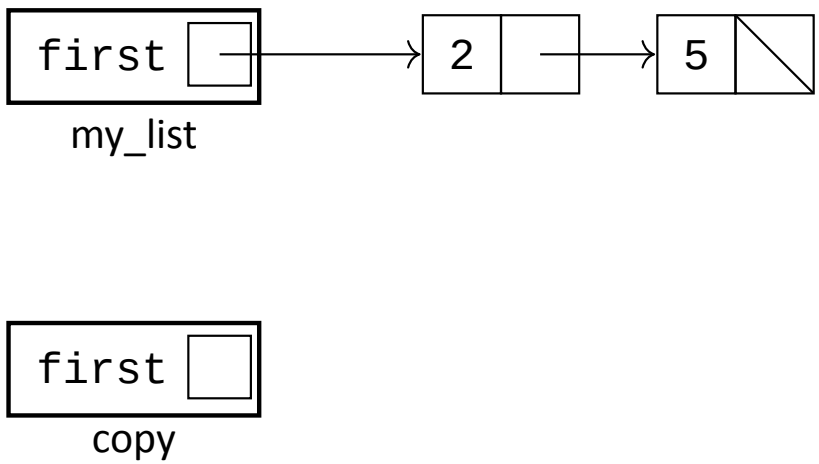
Constructors & destructors

```
{
  List my_list{}; // the constructor is called

  // do things with the list

} // the destructor is called
```

# Special Member Functions

## Copies

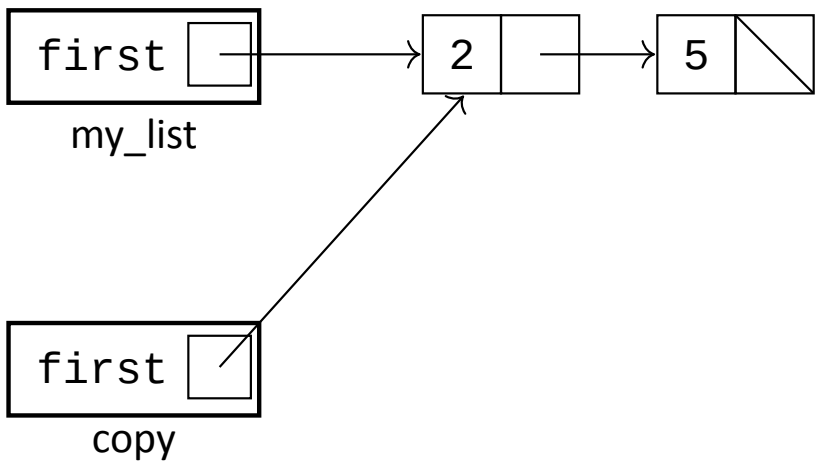
```
int main()
{
  List my_list{};
  my_list.insert(5);
  my_list.insert(2);
  {
    // copy my_list into copy
    List copy {my_list};
  }
}
```



# Special Member Functions

## Copies

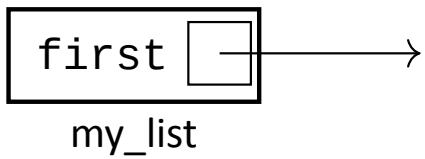
```
int main()
{
  List my_list{};
  my_list.insert(5);
  my_list.insert(2);
  {
    // copy my_list into copy
    List copy {my_list};
  }
}
```



# Special Member Functions

## Copies

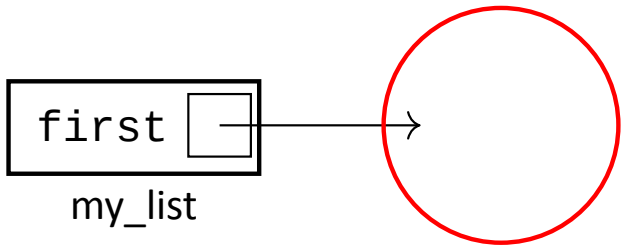
```
int main()
{
  List my_list{};
  my_list.insert(5);
  my_list.insert(2);
  {
    // copy my_list into copy
    List copy {my_list};
  }
}
```



# Special Member Functions

## Copies

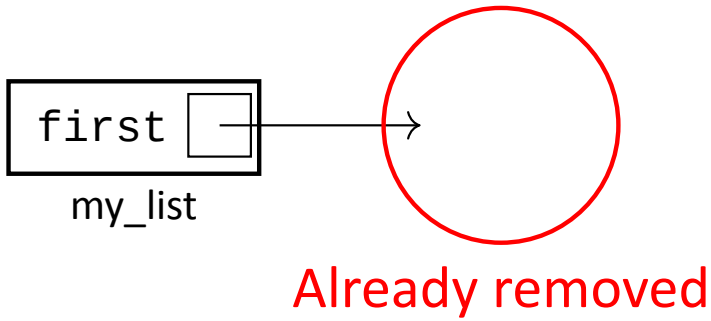
```
int main()
{
  List my_list{};
  my_list.insert(5);
  my_list.insert(2);
  {
    // copy my_list into copy
    List copy {my_list};
  }
}
```



# Special Member Functions

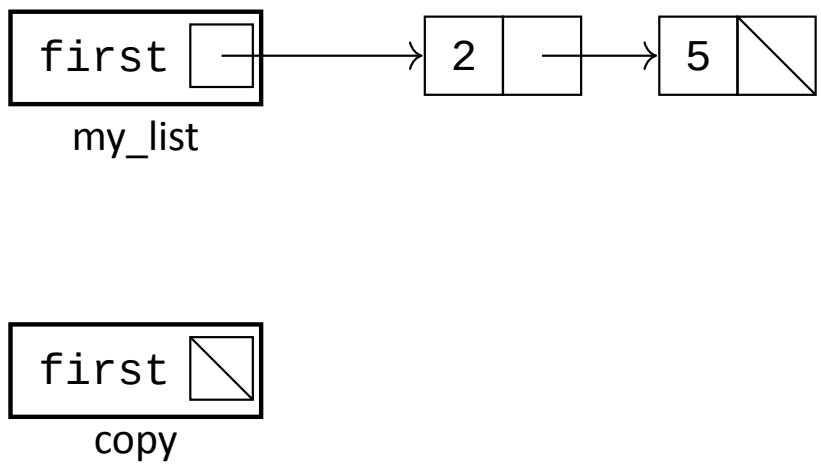
## Copies

```
int main()
{
  List my_list{};
  my_list.insert(5);
  my_list.insert(2);
  {
    // copy my_list into copy
    List copy {my_list};
  }
}
```



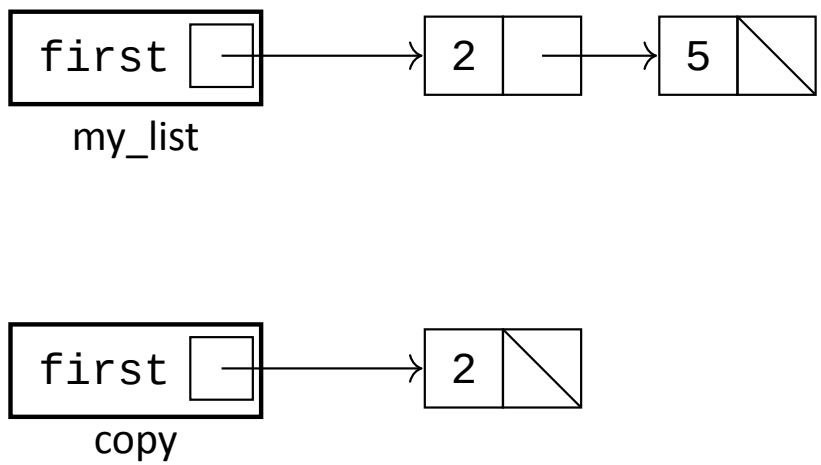
# Special Member Functions

Deep copies



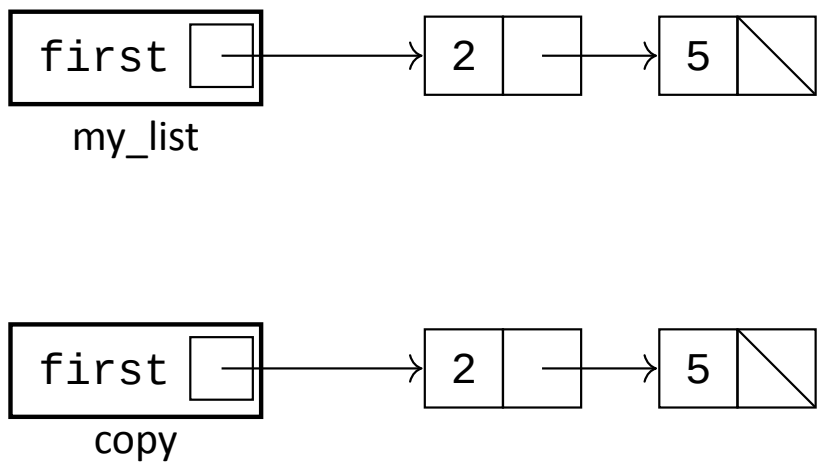
# Special Member Functions

Deep copies



# Special Member Functions

Deep copies



# Special Member Functions

## Copy constructor

```
class List
{
public:

  List(); // default constructor

  List(List const& other)
  {
    // perform a deep copy of the lists
  }

  ~List(); // destructor
  void remove();
  void insert(int value);

private:
  Node* first{};
};
```

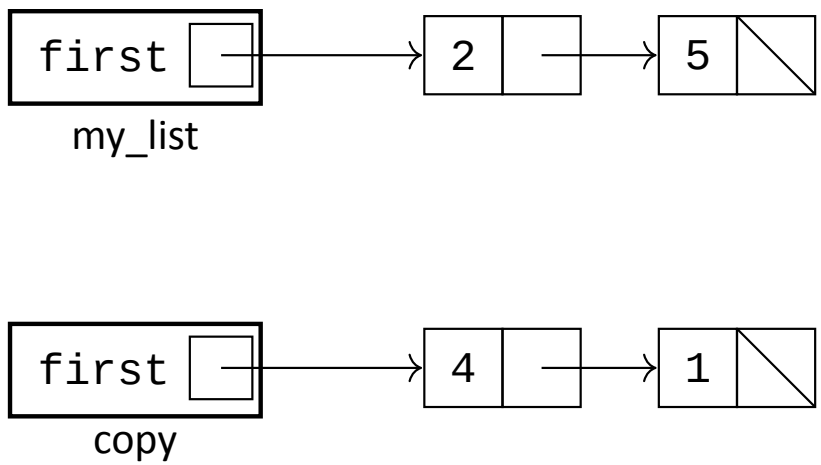
# Special Member Functions

Copy assignment

```
List my_list{};
my_list.insert(5);
my_list.insert(2);

List copy{};
copy.insert(1);
copy.insert(4);

// copy assignment
copy = my_list;
```

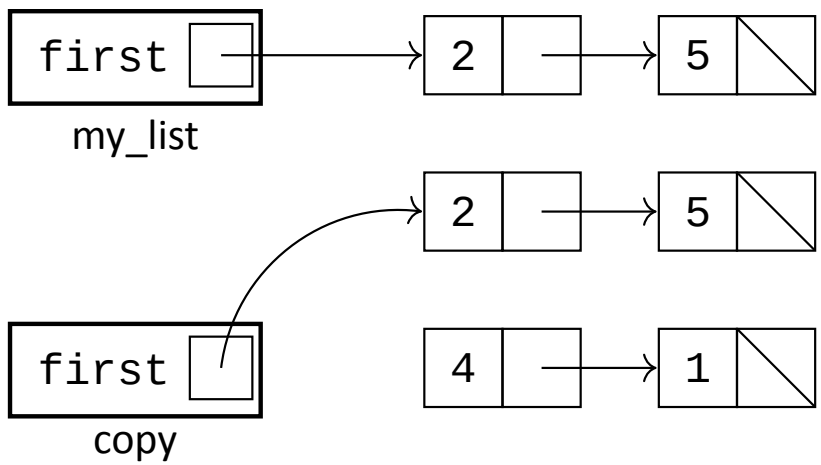
# Special Member Functions

Copy assignment



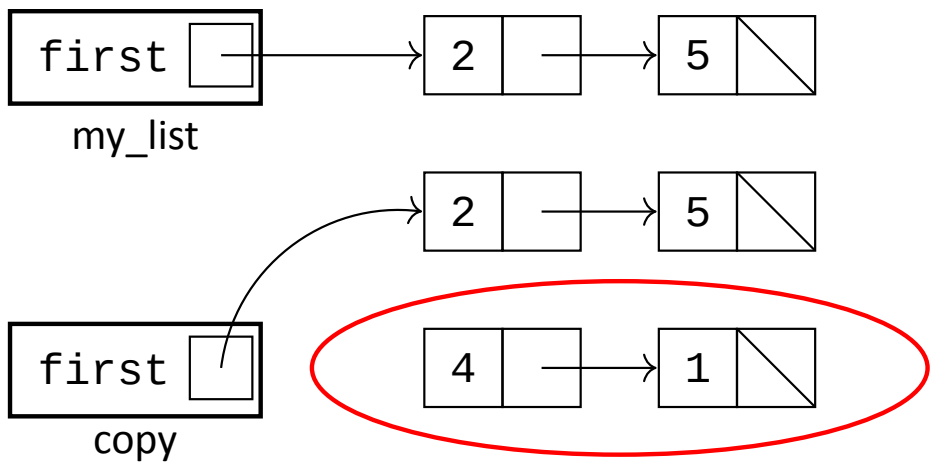
# Special Member Functions

Copy assignment



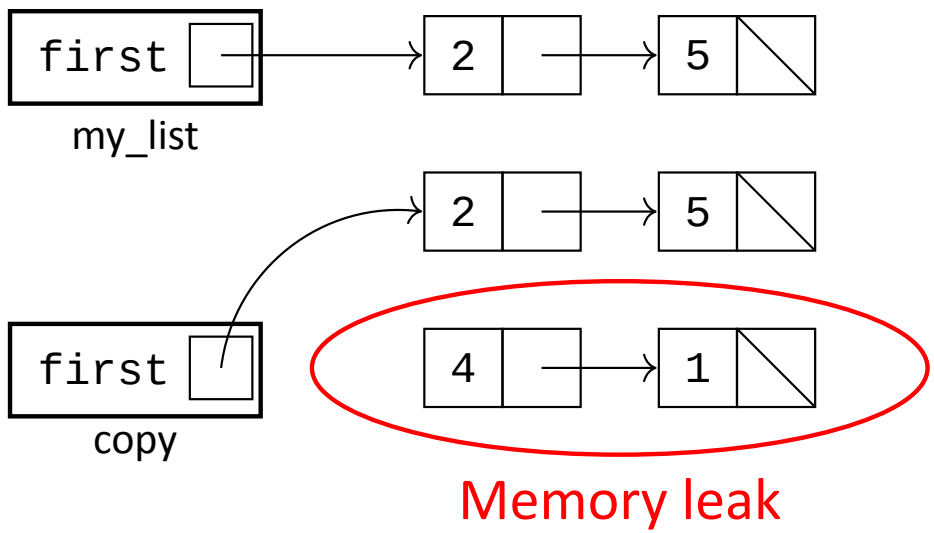
# Special Member Functions

Copy assignment



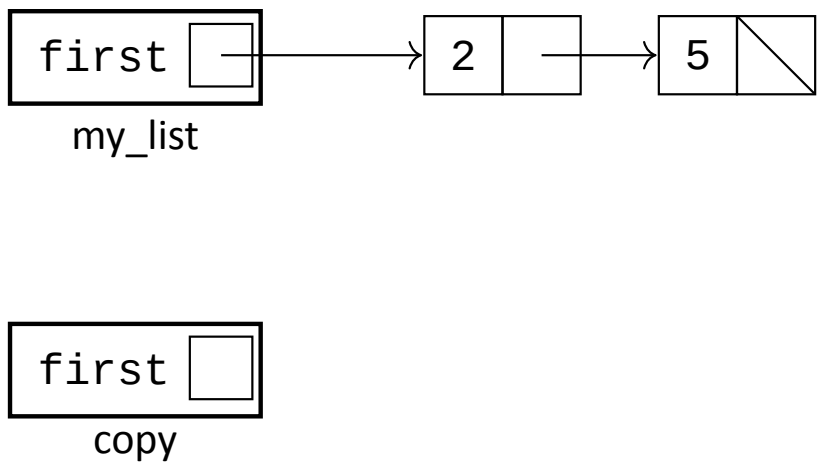
# Special Member Functions

Copy assignment



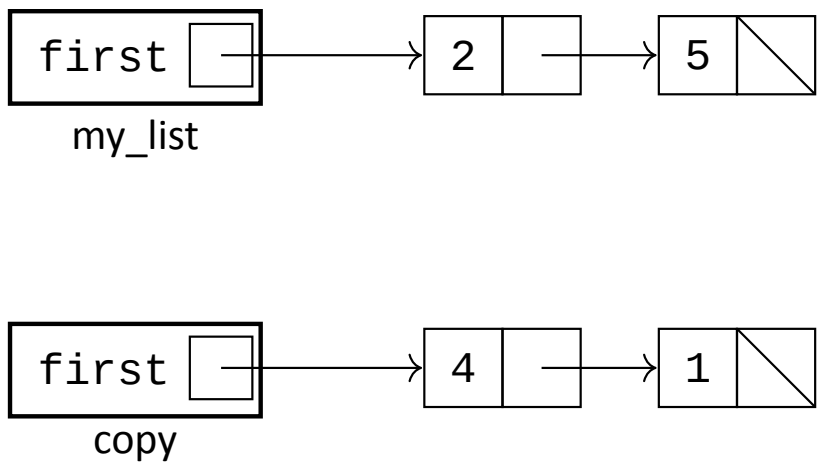
# Special Member Functions

Copy assignment



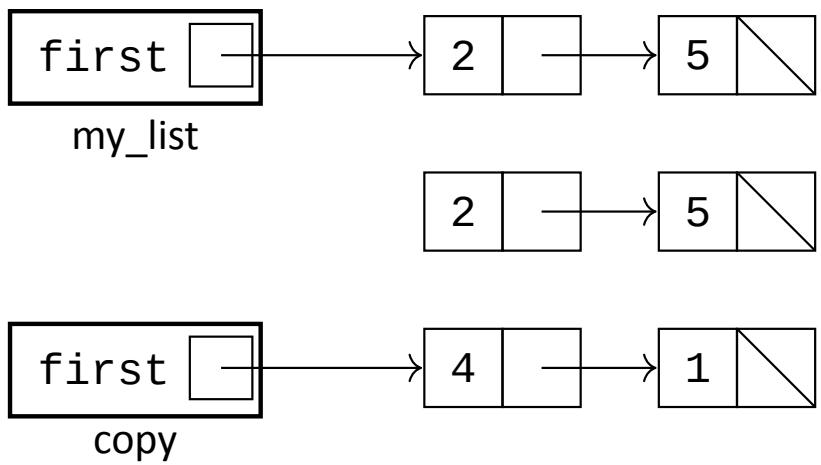
# Special Member Functions

Copy assignment



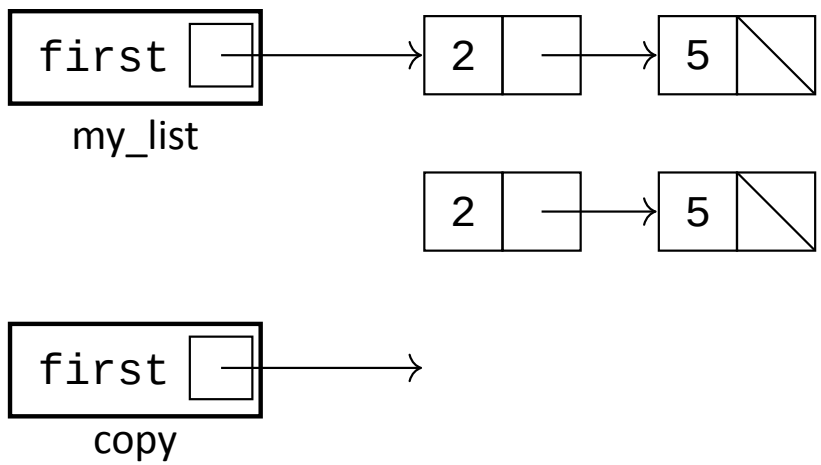
# Special Member Functions

Copy assignment



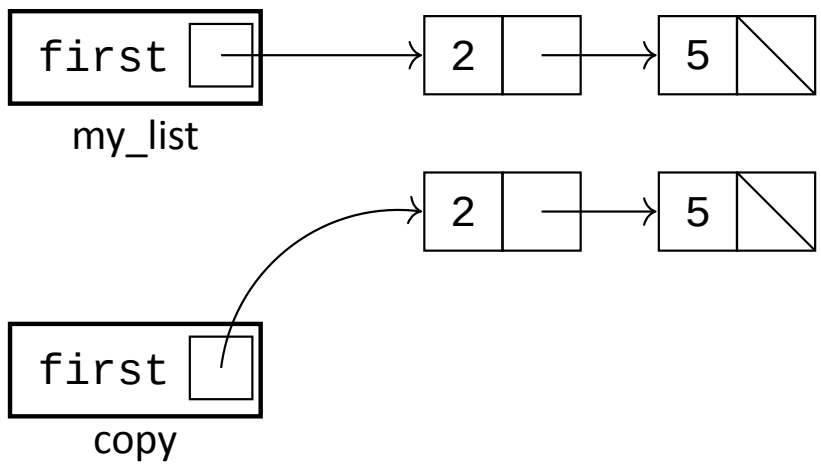
# Special Member Functions

Copy assignment



# Special Member Functions

Copy assignment



# Special Member Functions

Copy assignment operator

```
class List
{
public:

  List(); // default constructor
  List(List const& other); // copy constructor
  ~List(); // destructor

  List& operator=(List const& other)
  {
    // remove previous list and deep copy other
    return *this;
  }

  // ...
};
```

# Special Member Functions

Temporary objects

```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list {get_list()};
```


my_list

# Special Member Functions

Temporary objects

```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list {get_list()};
```

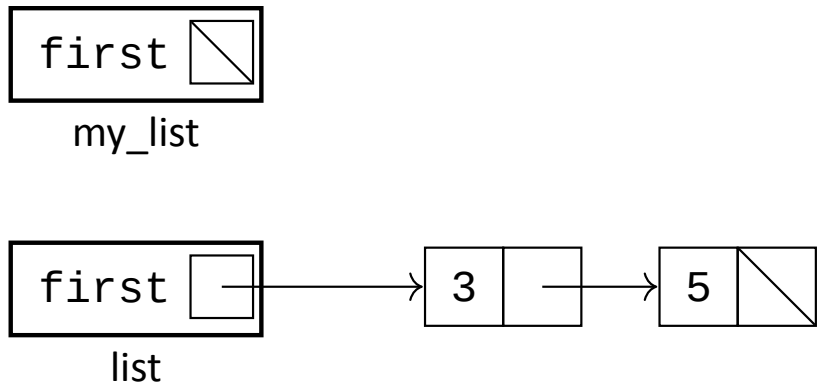

my_list


list

# Special Member Functions

Temporary objects

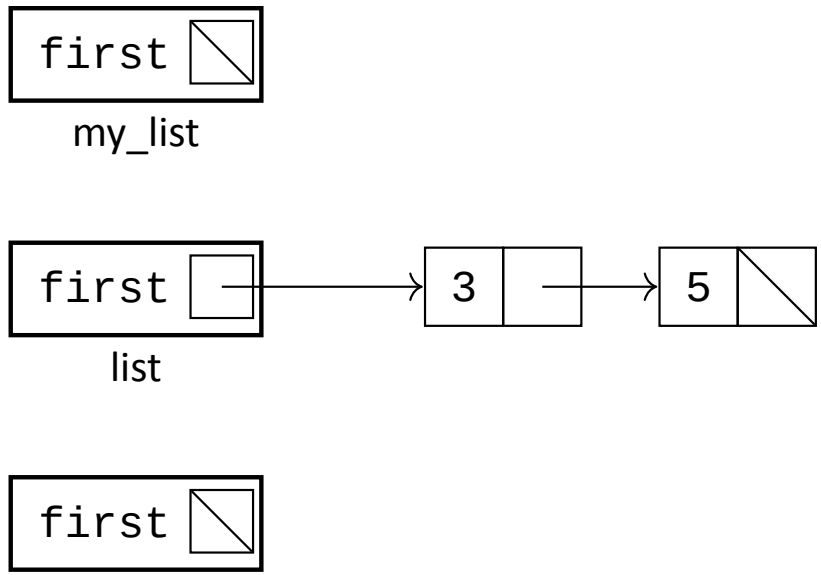
```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list {get_list()};
```



# Special Member Functions

Temporary objects

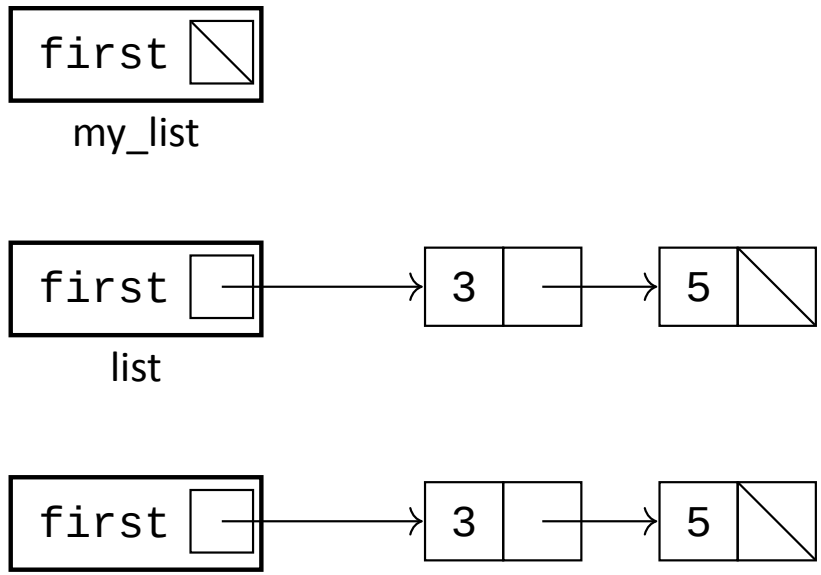
```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list {get_list()};
```



# Special Member Functions

Temporary objects

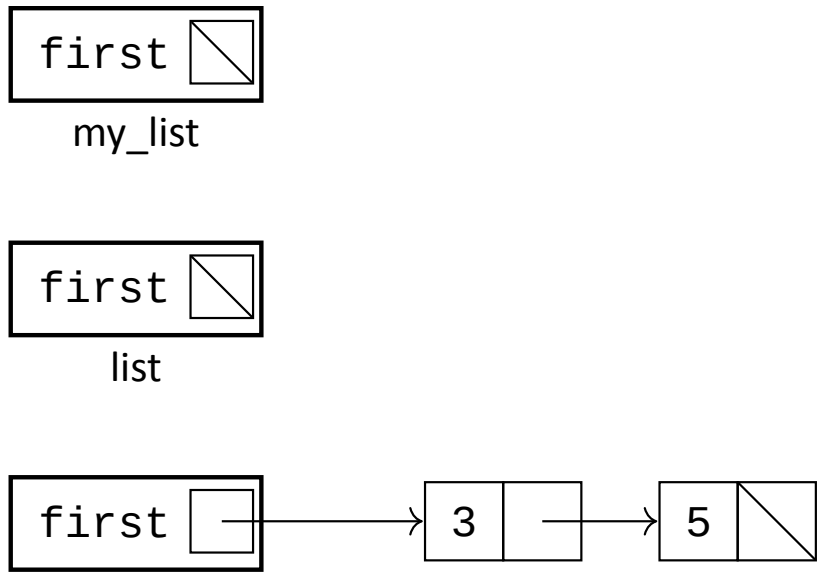
```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list {get_list()};
```



# Special Member Functions

Temporary objects

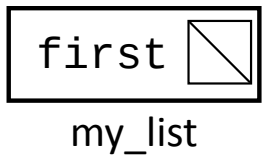
```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list {get_list()};
```

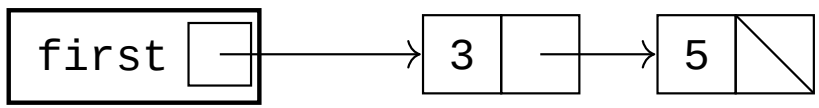


# Special Member Functions

Temporary objects

```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list {get_list()};
```

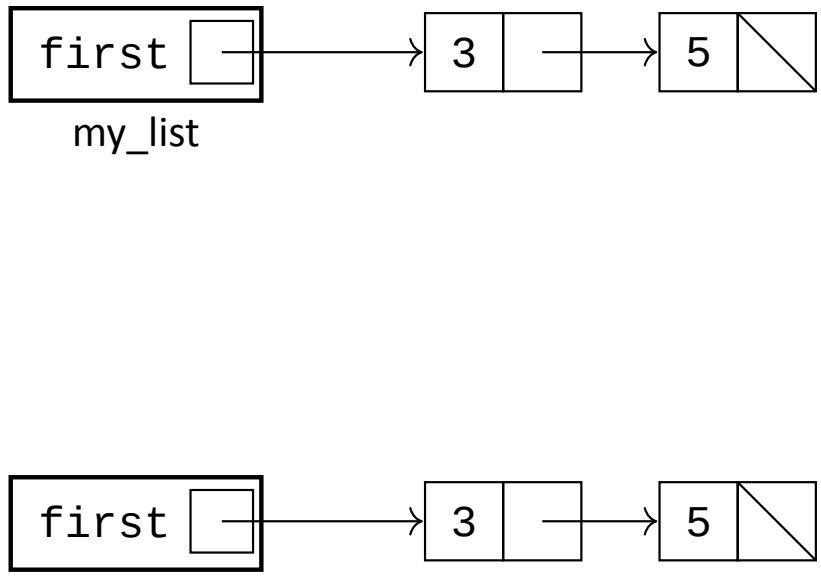


# Special Member Functions

Temporary objects

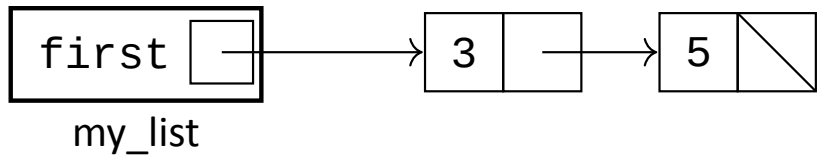
```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list {get_list()};
```



# Special Member Functions

Temporary objects

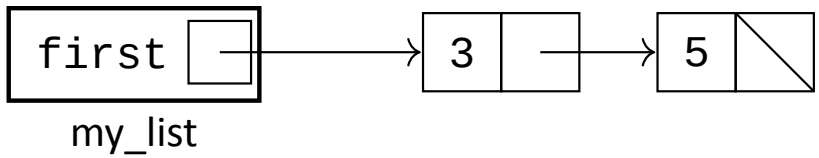
```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list {get_list()};
```




# Special Member Functions

Temporary objects

```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list {get_list()};
```



# Special Member Functions

Temporary objects

```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list {get_list()};
```


my_list

# Special Member Functions

Temporary objects

```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list {get_list()};
```

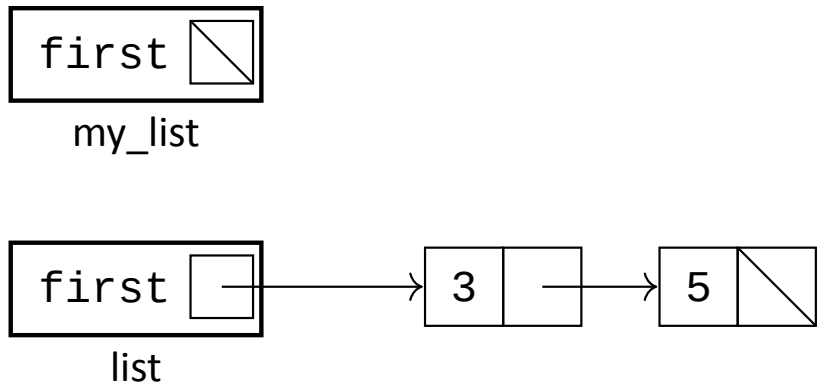

my_list


list

# Special Member Functions

Temporary objects

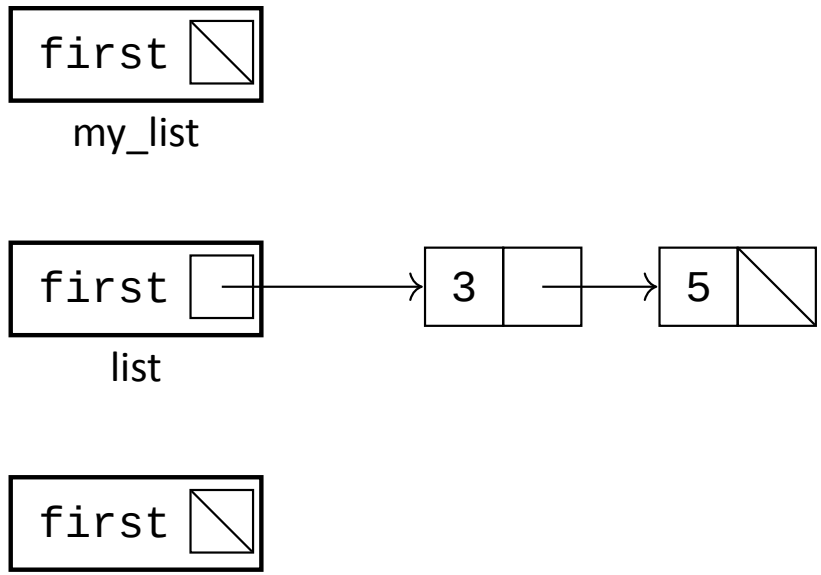
```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list {get_list()};
```



# Special Member Functions

Temporary objects

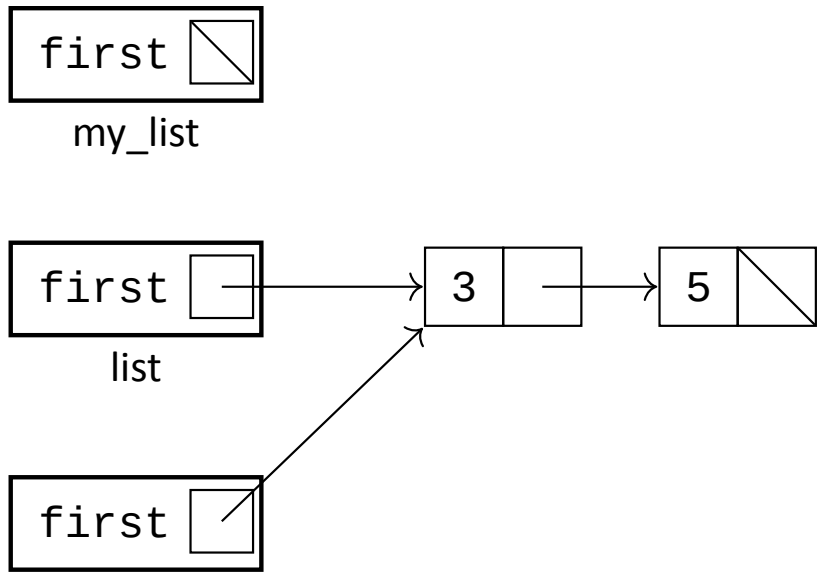
```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list {get_list()};
```



# Special Member Functions

Temporary objects

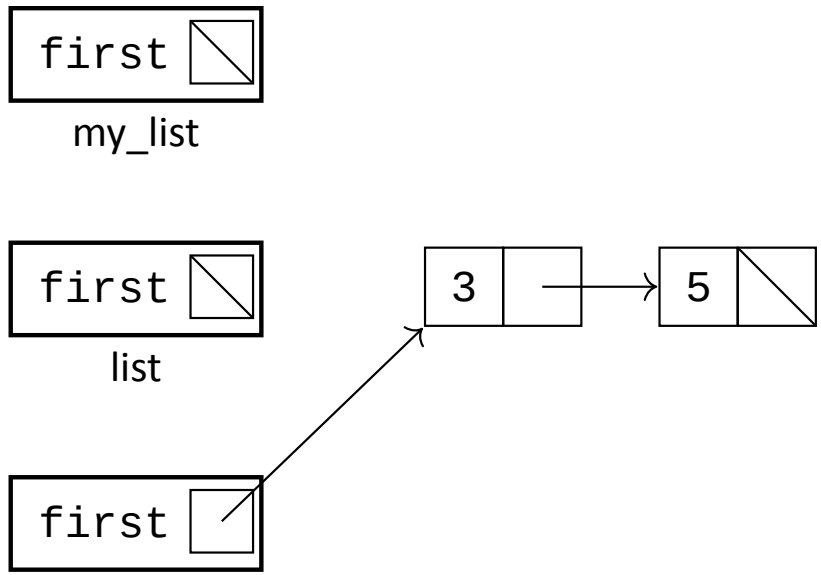
```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list {get_list()};
```



# Special Member Functions

Temporary objects

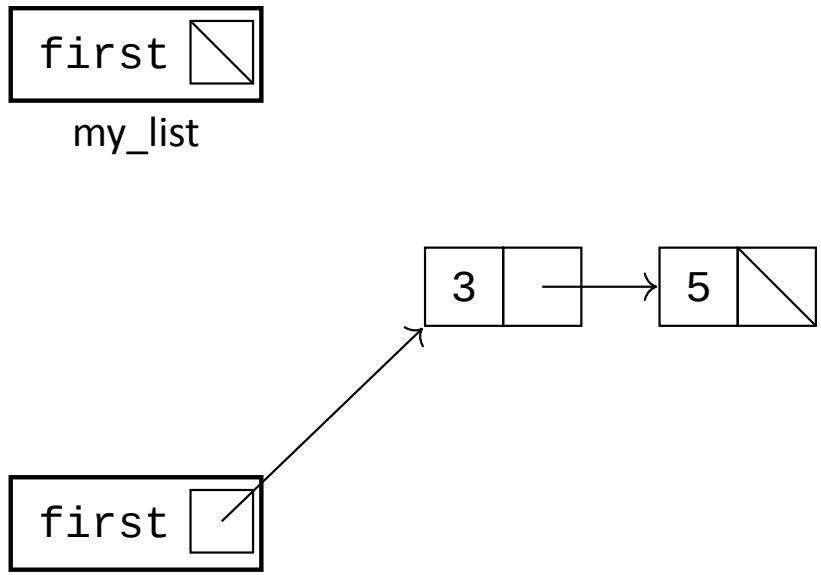
```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list {get_list()};
```



# Special Member Functions

Temporary objects

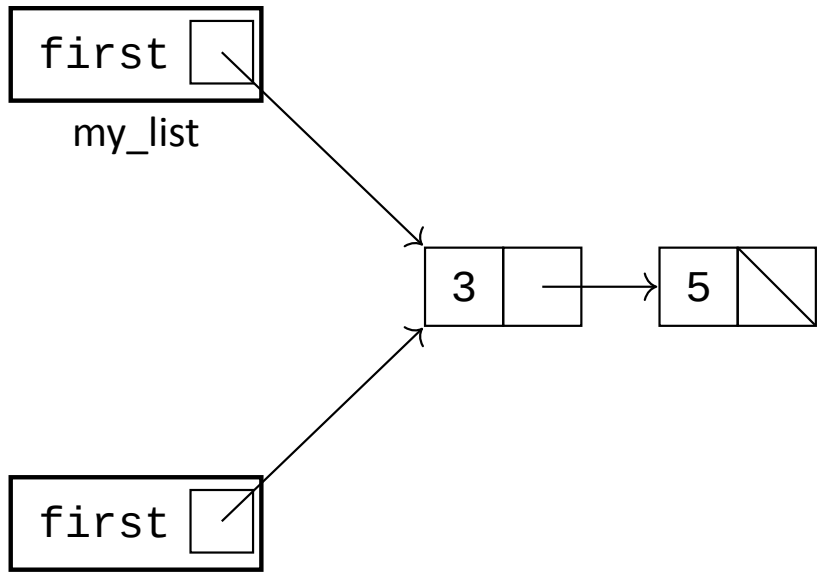
```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list {get_list()};
```



# Special Member Functions

Temporary objects

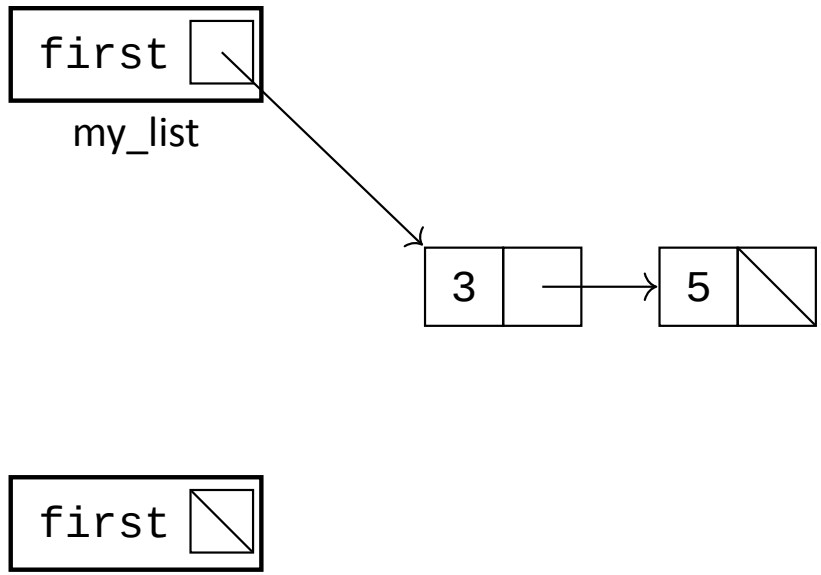
```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list {get_list()};
```



# Special Member Functions

Temporary objects

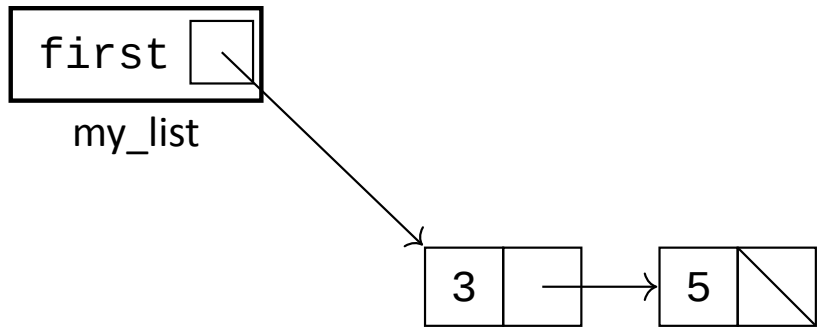
```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list {get_list()};
```



# Special Member Functions

Temporary objects

```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list {get_list()};
```



# Special Member Functions

Move constructor

```
class List
{
public:

  List(); // default constructor
  List(List const& other); // copy constructor

  List(List&& other)
  {
    // perform the move by shuffling the first pointers
  }

  ~List(); // destructor

  List& operator=(List const& other); // copy assignment operator

  // ...
};
```

# Special Member Functions

Move assignment

```
List get_list()
{
  List list{};
  list.insert(5);
  list.insert(3);
  return list;
}

List my_list{};
my_list.insert(4);
my_list.insert(2);

my_list = get_list();
```

# Special Member Functions

Move assignment operator

```
class List
{
public:

  List(); // default constructor
  List(List const& other); // copy constructor
  List(List&& other); // move constructor

  ~List(); // destructor

  List& operator=(List const& other); // copy assignment operator

  List& operator=(List&& other)
  {
    // remove old content of the list
    // move content from other to this object
  }

  // ...
};
```

# Special Member Functions

## Special Member Functions

```
class List
{
public:

  List(); // default constructor
  List(List const& other); // copy constructor
  List(List&& other); // move constructor
  ~List(); // destructor
  List& operator=(List const& other); // copy assignment operator
  List& operator=(List&& other); // move assignment operator

  // ...
};
```

# Special Member Functions

Nice initialization

```
List list1 {1, 2, 3};
List list2 {4, 5};
List list3 {6, 7, 8, 9};
```

```
#include <initializer_list>

class List
{
  public:
  List(std::initializer_list<int> list)
  {
    for (int i : list)
    {
      insert(i);
    }
  }
};
```

www.liu.se

LINKÖPING UNIVERSITY