CS 225

Data Structures

February 12 – Templates and Linked Memory

G Carl Evans

Method Dispatch

1) **Look at the type the method is called on**
2) **Look for the method in that type if found**
    1) **If type is virtual use runtime type and goto 2 ignoring virtual from now on**
    2) **Use method**
3) **No method found change to base type and goto 2**

Cube.cpp

```
1  Cube::print_1() {
2    cout << "Cube" << endl;
3  }
4
5  Cube::print_2() {
6    cout << "Cube" << endl;
7  }
8
9  virtual Cube::print_3() {
10   cout << "Cube" << endl;
11 }
12
13 virtual Cube::print_4() {
14   cout << "Cube" << endl;
15 }
16
17 // In .h file:
18 virtual print_5() = 0;
19
20
21
22
```

RubikCube.cpp

```
1  // No print_1() in RubikCube.cpp
2
3
4
5  RubikCube::print_2() {
6    cout << "Rubik" << endl;
7  }
8
9  // No print_3() in RubikCube.cpp
10
11
12
13 RubikCube::print_4() {
14   cout << "Rubik" << endl;
15 }
16
17 RubikCube::print_5() {
18   cout << "Rubik" << endl;
19 }
20
21
22
```

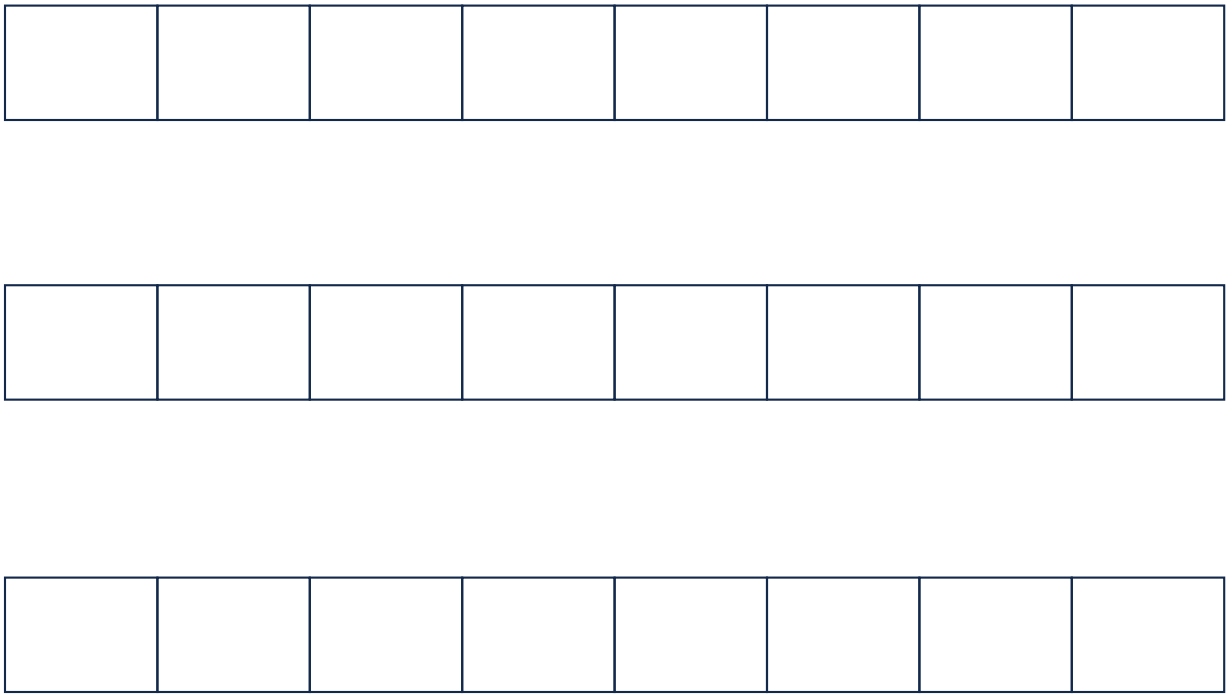
Runtime of Virtual Functions

virtual-main.cpp	`Cube c;`	`RubikCube c;`	`RubikCube rc;` `Cube &c = rc;`
`c.print_1();`			
`c.print_2();`			
`c.print_3();`			
`c.print_4();`			
`c.print_5();`			

Pure Virtual

List ADT

What types of “stuff” do we want in our list?



Templates

template1.cpp

```
1
2
3  T maximum(T a, T b) {
4    T result;
5    result = (a > b) ? a : b;
6    return result;
7  }
```

List.h

```
1  #pragma once
2
3
4  class List {
5    public:
6
7
8
9
10
11
12
13
14   private:
15
16
17
18 };
19
20
21
22
```

List.hpp

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
```

List Implementations

1.

2.

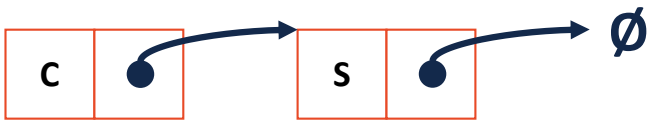
Linked Memory

List.h

```
28 class ListNode {
29   T & data;
30   ListNode * next;
31   ListNode(T & data) : data(data), next(NULL) { }
32 };
```

Linked Memory

Linked Memory



List.h

```
 1  #pragma once
 2
 3  template <class T>
 4  class List {
 5    public:
…      /* ... */
28    private:
29      class ListNode {
30        T & data;
31        ListNode * next;
32        ListNode(T & data) :
           data(data), next(NULL) { }
33      };
34
35
36
37
38
39  };
40
41
```

List.hpp

```
 1  #include "List.h"
 2
 3  template <class T>
 4  void List<T>::insertAtFront(const T& t) {
 5
 6
 7
 8
 9
10
11
12
13
14
15
16
17
18
19
20
21
22  }
```

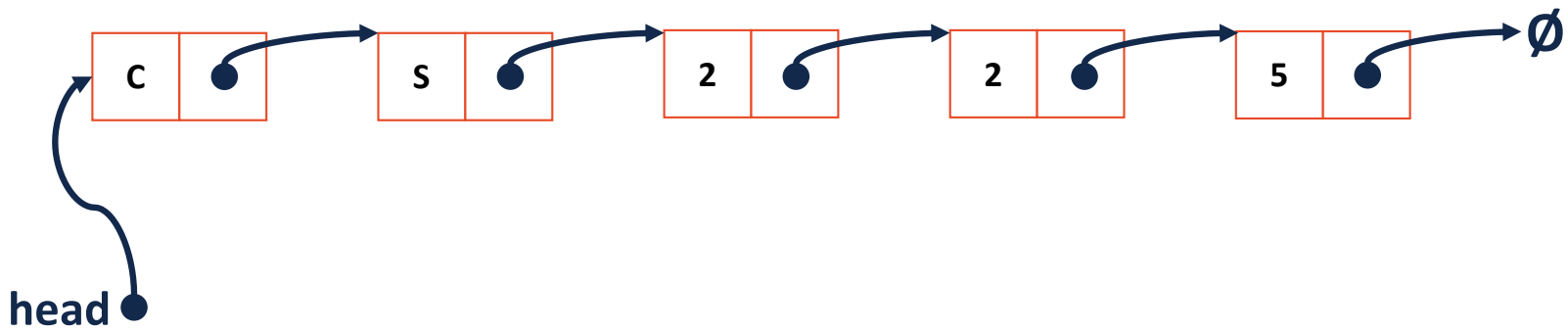
Running Time of Linked List `insertAtFront`

```
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
```

List.cpp

```
14  void List<T>::printReverse()
    const {
15
16
17
18
19
20
21
22  }
```

Linked Memory



Running Time of Linked List `printReverse`

List.cpp

```
24 template <typename T>
25 T List<T>::operator[](unsigned index) {
26
27
28
29
30
31 }
```

List.cpp

```
33 ListNode *& List<T>::_index(int index) const {
34
35
36
37
38
39
40 }
```