Compilation

0368-3133

Tutorial 11:

(More) Code Generation

Reminder

- **Last tutorial:**
  - Functions
  - Integers and global variables

- **Today:** all about pointers
  - Strings
  - Arrays
  - Classes

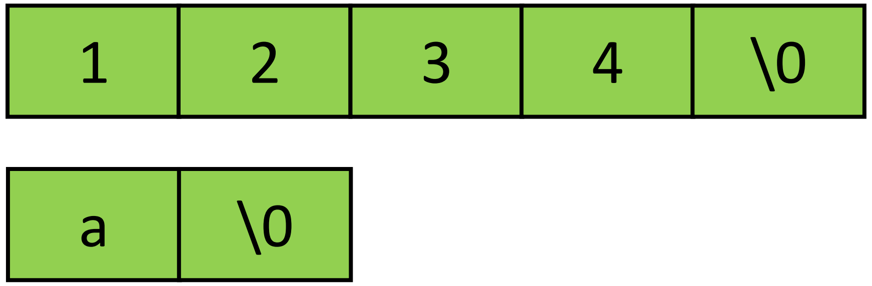
Reference Variables

- Variables (local or global) of type string, array or class store a pointer (memory address) to the data or nil (value 0)
- The data can be stored in:
  - The heap
  - Global data (only constant strings)
- Memory assignment translation is identical to integers

Strings

- We use null terminated strings
- Every character is one byte

```
string s1 = “1234”;
string s2 = “a”;
...
...
```



Strings

- Constant assignment

```
t0 = "1234"
```

$$\Downarrow$$

```
                      .data
la $t0, str_const     str_const: .asciiz "1234"
```

$\underbrace{\texttt{la \$t0, str\_const}}_{\text{goes to }\textbf{code section}}$ $\qquad$ $\underbrace{\texttt{.data str\_const: .asciiz "1234"}}_{\text{must be defined in }\textbf{data section}}$

Strings

- Global variable initialization

```
string z := “1234”;
```

$\Downarrow$

```
.data
z_str: .asciiz “1234”
z: .word z_str
```

Strings: Equality Comparison

- Assume that s1 and s2 are strings

```
if (s1 == s2) {
  ...
}
```

```
t1 = s1;
t2 = s2;
t3 = str_eq t1, t2
beq t3, 0, label
...
```

IR

Strings: Equality Comparison

- Inline string comparison

```
t1 = s1;
t2 = s2;
t3 = str_eq t1, t2
beq t3, 0, label
...
```

```
li $t3, 1 // result
move $s0, $t1
move $s1, $t2
str_eq_loop:
lb $s2, 0($s0)
lb $s3, 0($s1)
bne $s2, $s3, neq_label
beq $s2, $zero, str_eq_end
addu $s0, $s0, 1
addu $s1, $s1, 1
j str_eq_loop
neq_label:
li $t3, 0
str_eq_end:
```

Strings: Equality Comparison

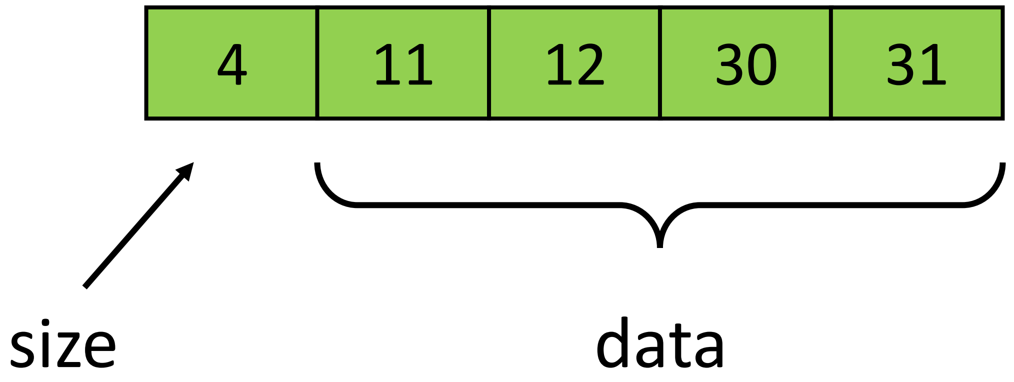
- Alternatively, create a function **str_eq**

```
t1 = s1;
t2 = s2;
t3 = str_eq t1, t2
beq t3, 0, label
...
```

```
subu $sp, $sp, 4
sw $t2, 0($sp)
subu $sp, $sp, 4
sw $t1, 0($sp)
jal str_eq
addu $sp, $sp, 8
move $t3, $v0
```

Arrays

- Each cell is 4 bytes (*int* or *pointer*)
- First cell is the **size** of the array
- The rest of the cells contain **data**



Arrays

- Creating arrays

```
t0 = new_array t1
```

```
li $v0, 9
move $a0, $t1
add $a0, $a0, 1
mul $a0, $a0, 4
syscall
move $t0, $v0
sw $t1, 0($t0)
```

Memory allocation

$a0 = number of *bytes* to allocate

Arrays

- Array access

```
t0 = array_access t1, t2
```

$\Rightarrow$

```
move $s0, $t2
add $s0, $s0, 1
mul $s0, $s0, 4
addu $s0, $t1, $s0
lw $t0, 0($s0)
```

Arrays

- Checking access violation

```
t0 = array_access t1, t2
```

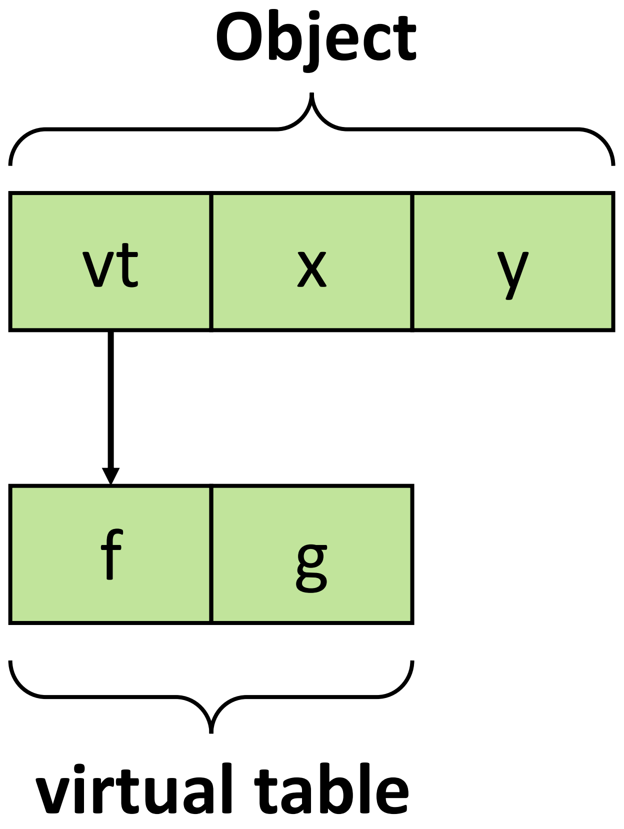
$\Rightarrow$

Branch on less than zero

```
bltz $t2, abort
lw $s0, 0($t1)
bge $t2, $s0, abort
move $s0, $t2
add $s0, $s0, 1
mul $s0, $s0, 4
addu $s0, $t1, $s0
lw $t0, 0($s0)
...
abort:
li $v0, 10
syscall
```

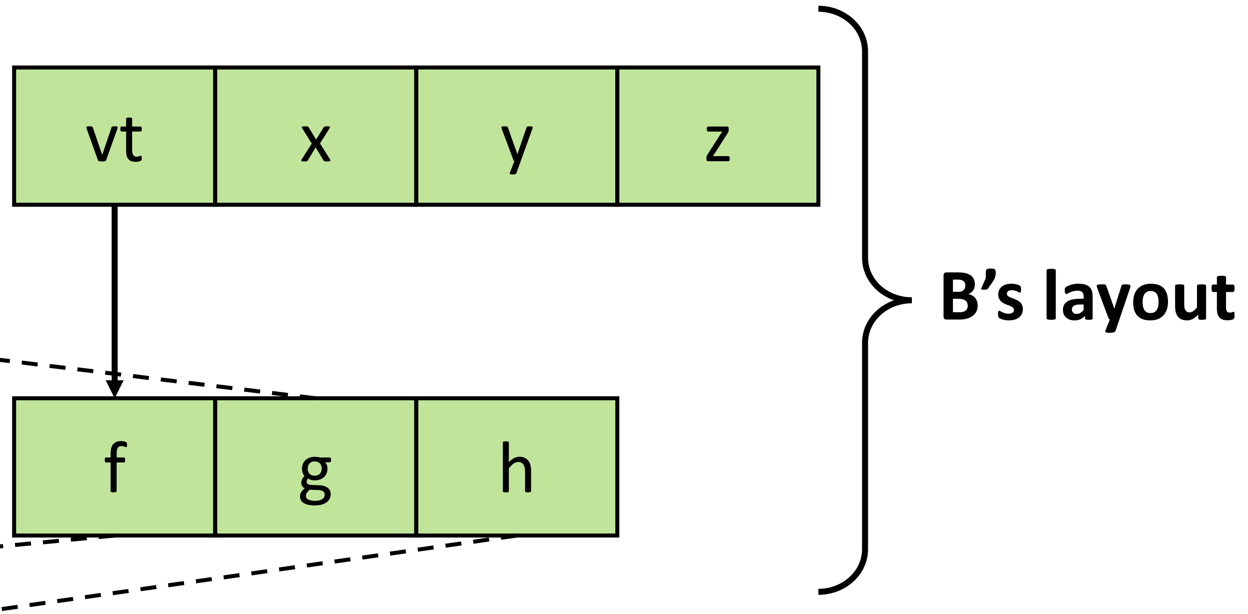
Class Layout

```
class A {
  int x;
  string y;
  int f() { …
  int g() { …
}
```



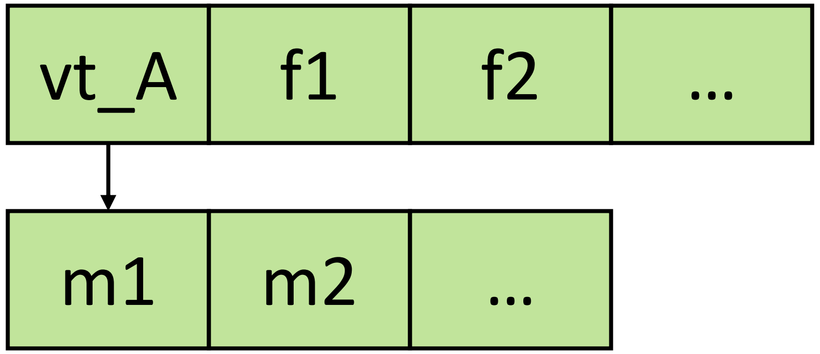
Class Layout

```
class A {
  int x;
  string y;
  int f() { …
  int g() { …
}
class B extends A {
  int z;
  int f() { …
  int h() { …
}
```



Creating Objects

```
class A {
  int f1 = c;
  …
  int m1() { …
  …
}

A a = new A;
```



Creating Objects

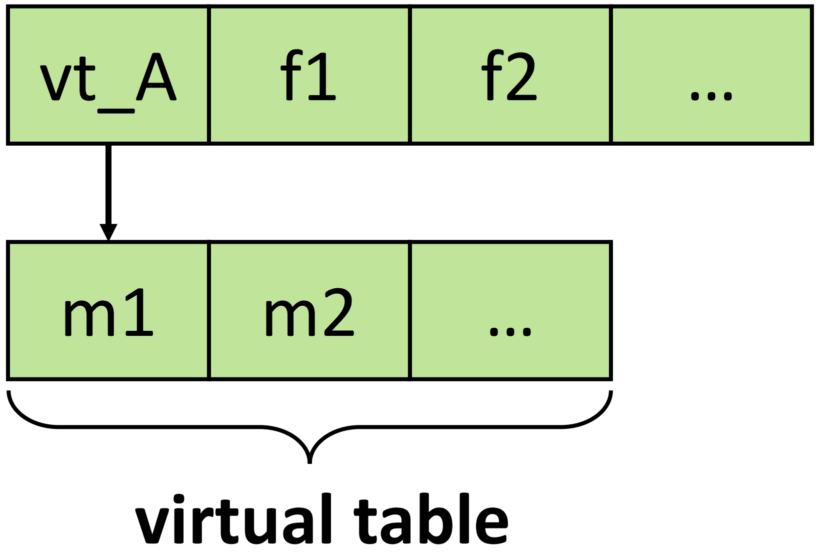
```
class A {
   int f1 = c;
   …
   int m1() { …
   …
}

A a = new A;
```

```
t0 = new_class A
a = t0
```

IR

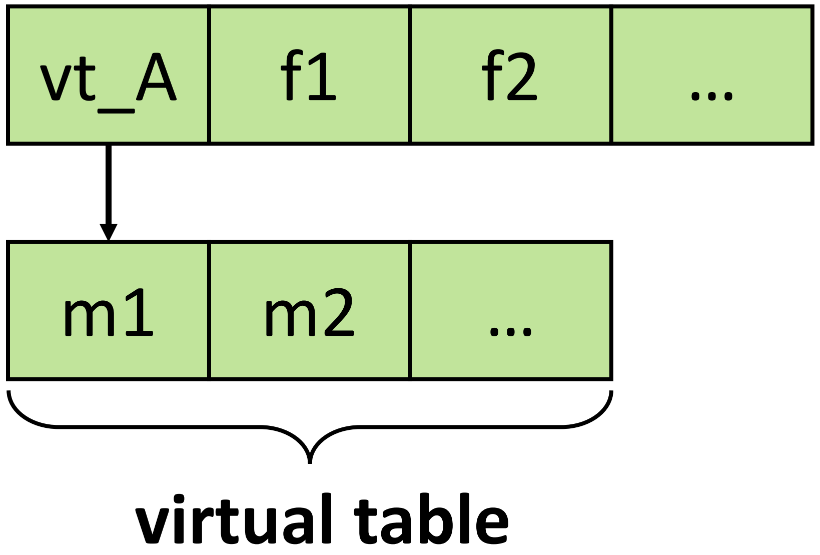
Creating Objects

```
t0 = new_class A
```



Creating Objects

```
t0 = new_class A
```

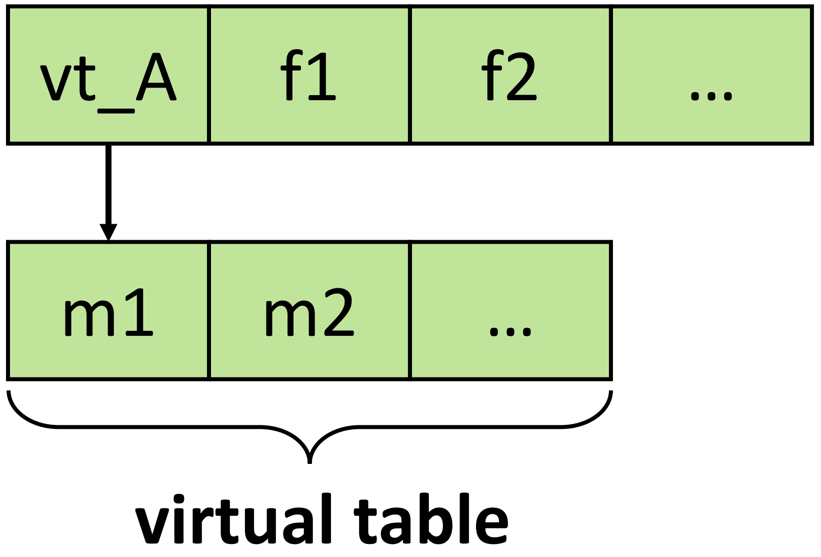


Generated once

```
.data
vt_A:
.word m1
.word m2
...

.text
m1:                 // method code
...
m2:                 // method code
...
```

Creating Objects

```
t0 = new_class A
```

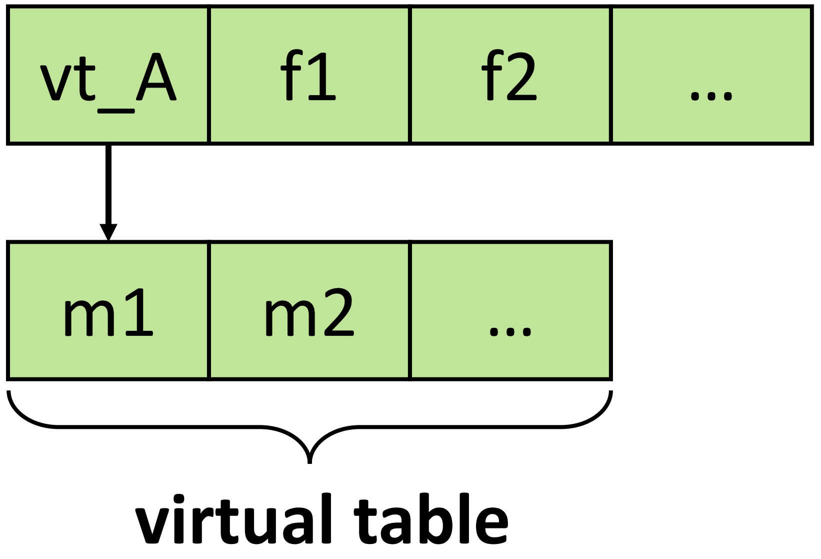


```
.data
vt_A:
.word m1
.word m2
...
```

```
.text
li $v0, 9
li $a0, size-of-A
syscall
move $t0, $v0
```

Creating Objects

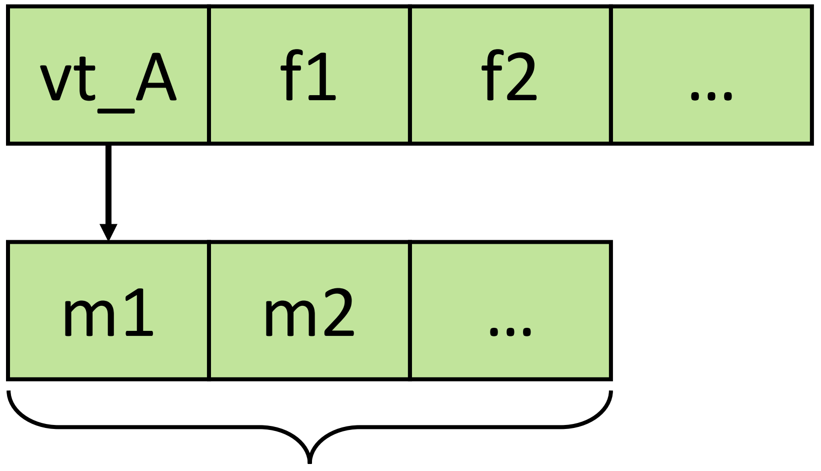
```
t0 = new_class A
```



```
.data
vt_A:
.word m1
.word m2
...

.text
li $v0, 9
li $a0, size-of-A
syscall
move $t0, $v0
la $s0, vt_A
sw $s0, 0($t0)
```

Creating Objects

```
t0 = new_class A
```



virtual table

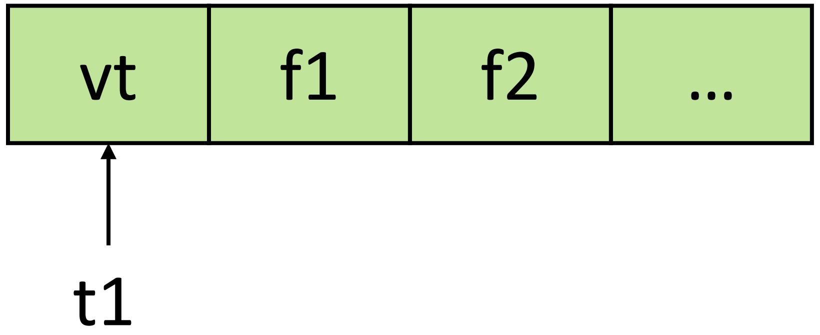
```
.data
vt_A:
.word m1
.word m2
...

.text
li $v0, 9
li $a0, size-of-A
syscall
move $t0, $v0
la $s0, vt_A
sw $s0, 0($t0)
li $s0, c
sw $s0, 4($t0)
...
```

Field Access

```
t0 = field_access t1, f
```

```
lw $t0, off($t1)
```

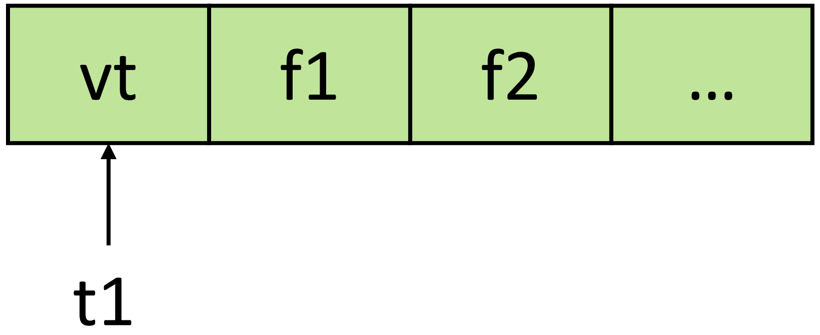
use annotated AST



Field Access

```
t0 = field_access t1, f
```

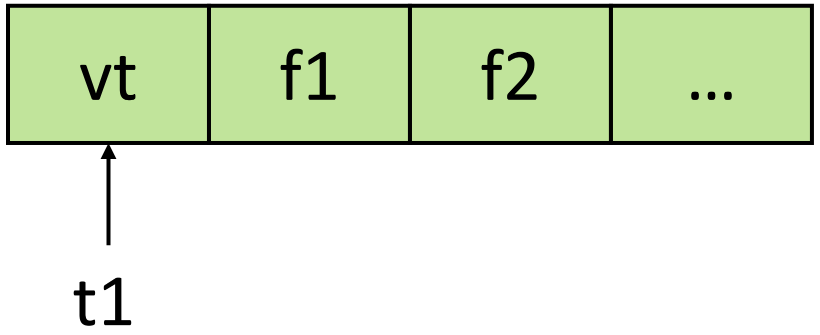
```
beq $t1, 0, abort
lw $t0, off($t1)
...
abort:
li $v0, 10
syscall
```



Field Access

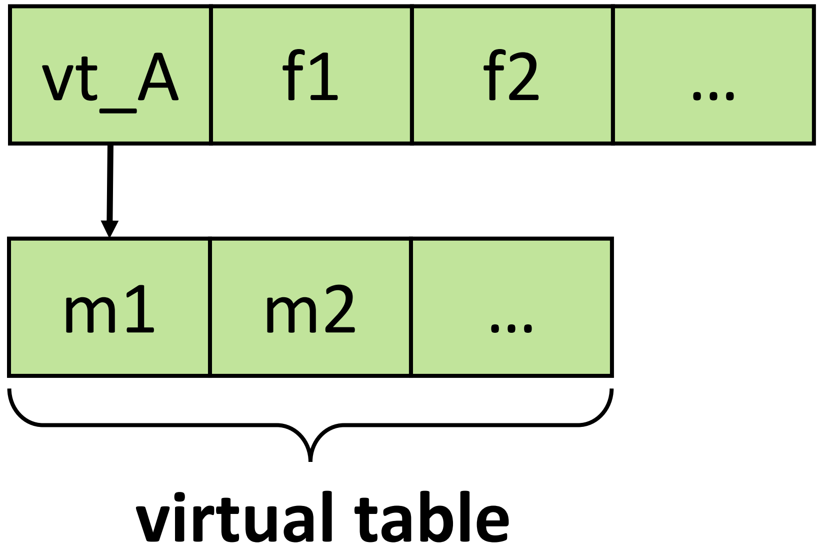
```
field_set t1, f, t2
```

```
beq $t1, 0, abort
sw $t2, off($t1)
...
abort:
li $v0, 10
syscall
```



Method Calls

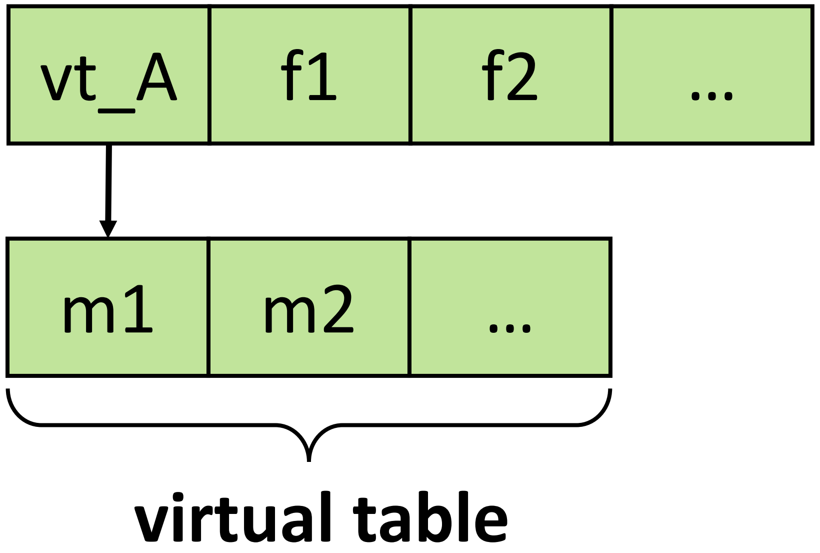
```
t2 = virtual_call t0, m, t1
```



Method Calls

```
t2 = virtual_call t0, m, t1
```

```
subu $sp, $sp, 4
sw $t1, 0($sp)
```

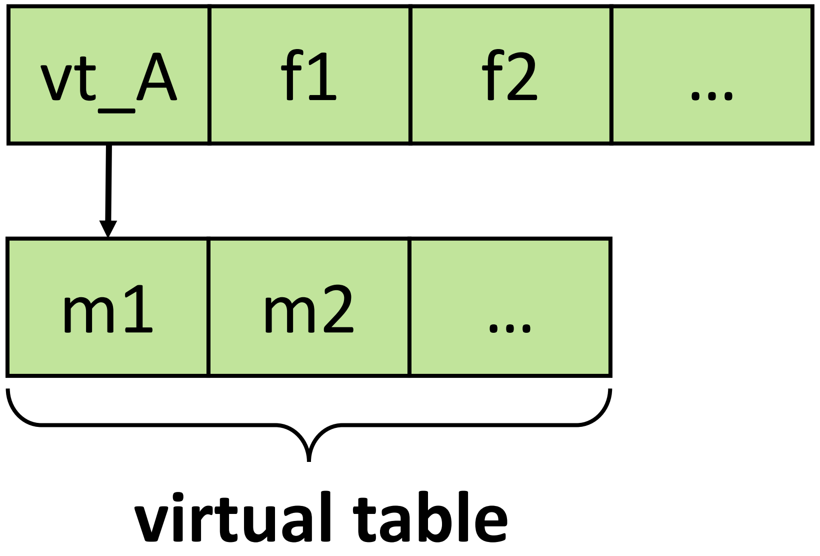


Method Calls

```
t2 = virtual_call t0, m, t1
```

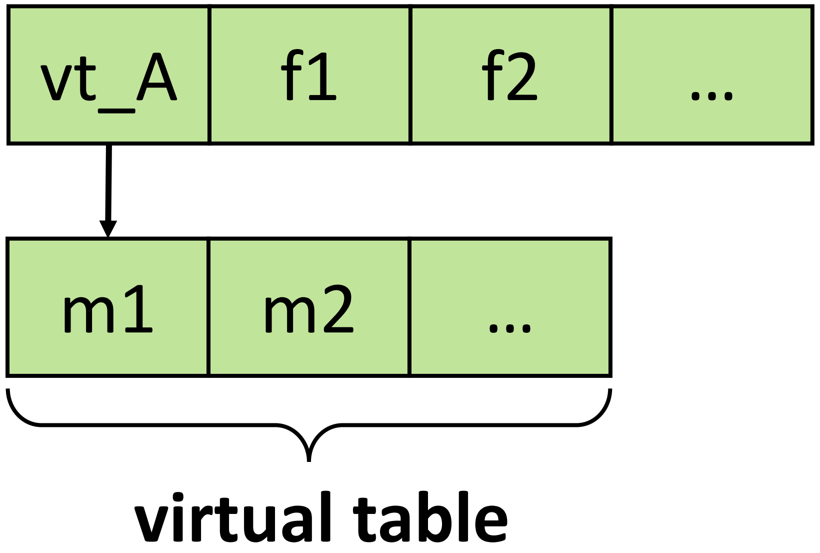
```
subu $sp, $sp, 4
sw $t1, 0($sp)
subu $sp, $sp, 4
sw $t0, 0($sp)
```

Used to access the object's members inside a method



Method Calls

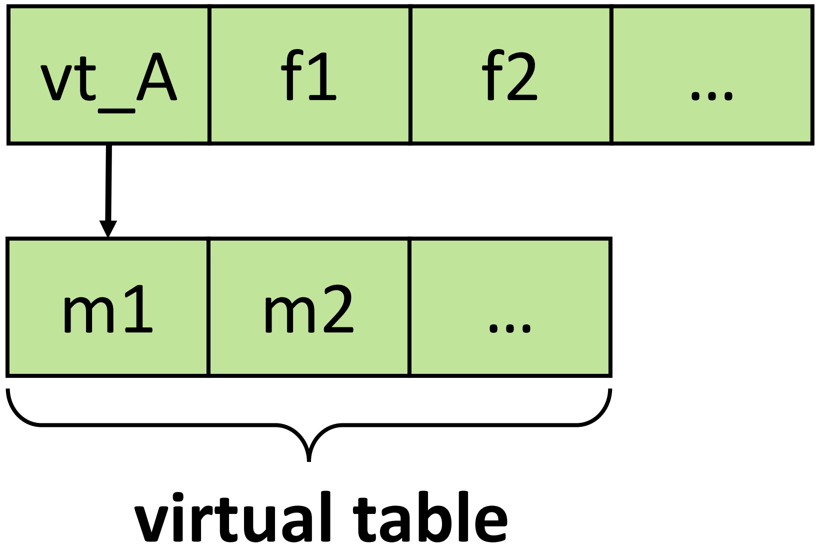
```
t2 = virtual_call t0, m, t1
```



```
subu $sp, $sp, 4
sw $t1, 0($sp)
subu $sp, $sp, 4
sw $t0, 0($sp)
lw $s0, 0($t0)
```

Method Calls

```
t2 = virtual_call t0, m, t1
```

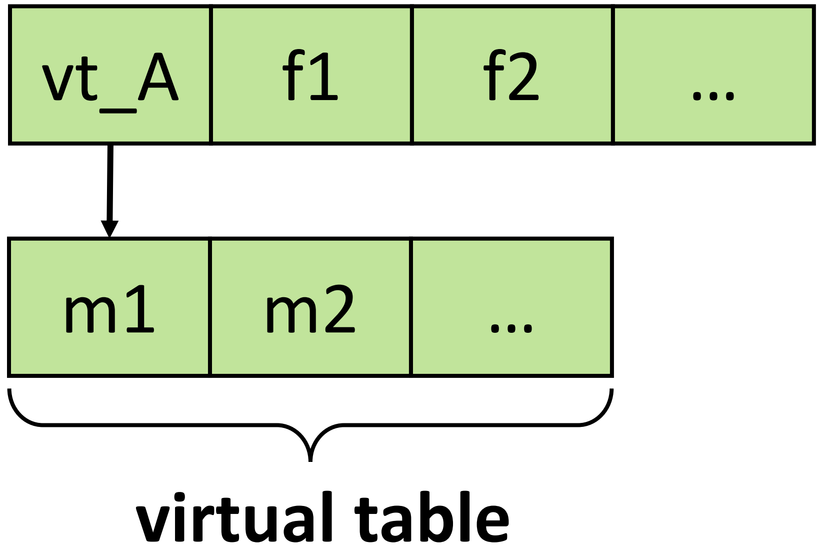


```
subu $sp, $sp, 4
sw $t1, 0($sp)
subu $sp, $sp, 4
sw $t0, 0($sp)
lw $s0, 0($t0)
lw $s1, off($s0)
```

use annotated AST

Method Calls

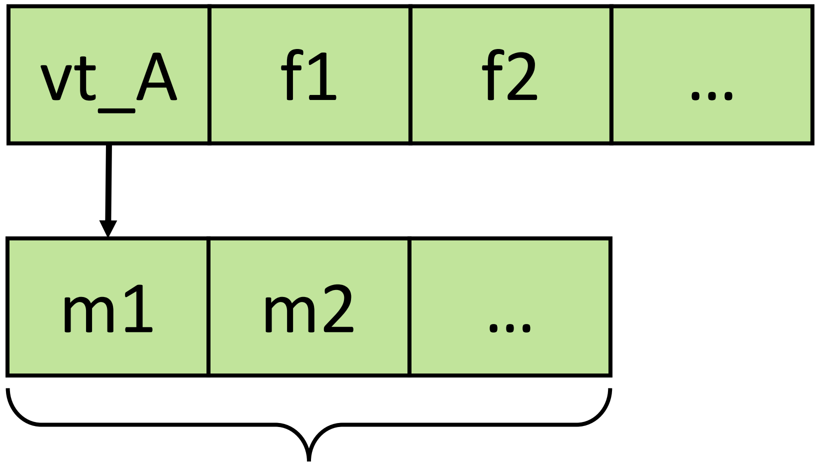
```
t2 = virtual_call t0, m, t1
```



```
subu $sp, $sp, 4
sw $t1, 0($sp)
subu $sp, $sp, 4
sw $t0, 0($sp)
lw $s0, 0($t0)
lw $s1, off($s0)
jalr $s1
```

Method Calls

```
t2 = virtual_call t0, m, t1
```

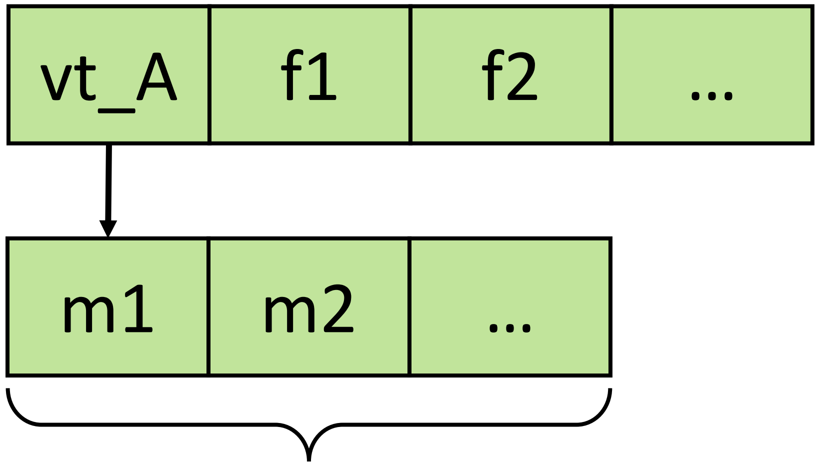


virtual table

```
subu $sp, $sp, 4
sw $t1, 0($sp)
subu $sp, $sp, 4
sw $t0, 0($sp)
lw $s0, 0($t0)
lw $s1, off($s0)
jalr $s1
addu $sp, $sp, 8
```

Method Calls

```
t2 = virtual_call t0, m, t1
```



virtual table

```
subu $sp, $sp, 4
sw $t1, 0($sp)
subu $sp, $sp, 4
sw $t0, 0($sp)
lw $s0, 0($t0)
lw $s1, off($s0)
jalr $s1
addu $sp, $sp, 8
move $t2, $v0
```

Method Calls

```
class A {
   void m1(int x) {}
   void m2(int x) {}
}
class B extends A {
   void m2(int x) {}
}

void main(){
   B b = new B;
   z = b.m2(7)
}
```

```
t0 = new_class B
b = t0
t1 = b
t2 = 7
t3 = virtual_call t1, m2, t2
z = t3
```

IR

Method Calls

```
t0 = new_class B
b = t0
t1 = b
t2 = 7
t3 = virtual_call t1, m2, t2
z = t3
```

```
.data
vt_B:
.word A_m1
.word B_m2
```

$\left.\vphantom{\begin{matrix}a\\a\\a\\a\end{matrix}}\right\}$ **data section**

Method Calls

```
t0 = new_class B
b = t0
t1 = b
t2 = 7
t3 = virtual_call t1, m2, t2
z = t3
```

```
li $v0, 9
li $a0, 4
syscall
move $t0, $v0
la $s0, vt_B
sw $s0, 0($t0)
```

```
.data
vt_B:
.word A_m1
.word B_m2
```

$\Big\}$ **data section**

Method Calls

```
t0 = new_class B
b = t0
t1 = b
t2 = 7
t3 = virtual_call t1, m2, t2
z = t3
```

```
li $v0, 9
li $a0, 4
syscall
move $t0, $v0
la $s0, vt_B
sw $s0, 0($t0)
sw $t0, off_b($fp)
```

```
.data
vt_B:
.word A_m1
.word B_m2
```

$\Big\}$ **data section**

Method Calls

```
t0 = new_class B
b = t0
t1 = b
t2 = 7
t3 = virtual_call t1, m2, t2
z = t3
```

```
li $v0, 9
li $a0, 4
syscall
move $t0, $v0
la $s0, vt_B
sw $s0, 0($t0)
sw $t0, off_b($fp)
lw $t1, off_b($fp)
```

```
.data
vt_B:
.word A_m1
.word B_m2
```

$\Big\}$ **data section**

Method Calls

```
t0 = new_class B
b = t0
t1 = b
t2 = 7
t3 = virtual_call t1, m2, t2
z = t3
```

```
li $v0, 9
li $a0, 4
syscall
move $t0, $v0
la $s0, vt_B
sw $s0, 0($t0)
sw $t0, off_b($fp)
lw $t1, off_b($fp)
li $t2, 7
```

```
.data
vt_B:
.word A_m1
.word B_m2
```

$\}$ **data section**

Method Calls

```
t0 = new_class B
b = t0
t1 = b
t2 = 7
t3 = virtual_call t1, m2, t2
z = t3
```

```
.data
vt_B:
.word A_m1
.word B_m2
```

data section

```
li $v0, 9
li $a0, 4
syscall
move $t0, $v0
la $s0, vt_B
sw $s0, 0($t0)
sw $t0, off_b($fp)
lw $t1, off_b($fp)
li $t2, 7
subu $sp, $sp, 4
sw $t2, 0($sp)
subu $sp, $sp, 4
sw $t1, 0($sp)
lw $s0, 0($t1)
lw $s1, 4($s0)
jalr $s1
addu $sp, $sp, 8
move $t3, $v0
```

Method Calls

```
t0 = new_class B
b = t0
t1 = b
t2 = 7
t3 = virtual_call t1, m2, t2
z = t3
```

```
.data
vt_B:
.word A_m1
.word B_m2
```

$\}$ **data section**

```
li $v0, 9
li $a0, 4
syscall
move $t0, $v0
la $s0, vt_B
sw $s0, 0($t0)
sw $t0, off_b($fp)
lw $t1, off_b($fp)
li $t2, 7
subu $sp, $sp, 4
sw $t2, 0($sp)
subu $sp, $sp, 4
sw $t1, 0($sp)
lw $s0, 0($t1)
lw $s1, 4($s0)
jalr $s1
addu $sp, $sp, 8
move $t3, $v0
sw $t3, off_z($fp)
```