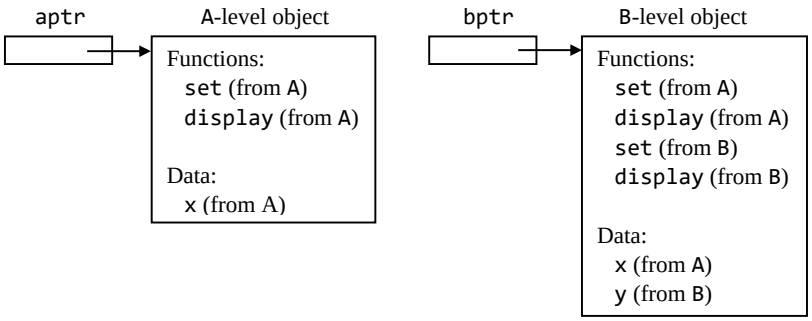
16 C++ Inheritance and Virtual Functions

Inheritance

```
aptr = new A;

bptr = new B;
```




```
class B: public A   // indicates B is derived from A
```

```
 1 // ex1601.cpp  Inheritance
 2 #include <iostream>
 3 using namespace std;
 4 class A
 5 {
 6    public:
 7       void set(int n);
 8       void display();
 9    protected:
10       int x;
11 };
12 void A::set(int n)
13 {
14    x = n;
15 }
16 void A::display()
17 {
18    cout << x << endl;
19 }
20 //===================
21 class B: public A
22 {
23    public:
24       void set(int n, int m);
25       void display();
26    private:
27       int y;
28 };
29 void B::set(int n, int m)
30 {
31    x = n;
32    y = m;
33 }
34 void B::display()
35 {
36    cout << x << " " << y << endl;
37 }
38 //==============================
39 int main()
40 {
41    A *aptr;
42    B *bptr;
43    aptr = new A;
44    aptr->set(1);
45    aptr->display();
46    bptr = new B;
47    bptr->set(2, 3);
48    bptr->display();
49    aptr = bptr;
50    aptr->display();
51    return 0;
52 }
```

Indicates B is derived from A

```
; ex1601.a  Inheritance
startup:  bl main
          halt
;==============================================================
                           ; #include <iostream>
                           ; using namespace std;
                           ; class A
                           ; {
                           ;    public:
                           ;       void set(int a);
                           ;       void display();
                           ;    protected:
                           ;       int x;
                           ; };
@A@set$i: push lr          ; A::set(int n)
          push fp          ; {
          mov fp, sp

          ldr r0, fp, 3    ;    x = n;
          ldr r1, fp, 2
          str r0, r1, 0

          mov sp, fp       ; }
          pop fp
          pop lr
          ret

@A@display$v:              ; void A::display()
          push lr          ; {
          push fp
          mov fp, sp

          ldr r0, fp, 2    ;    cout << x << endl;
          ldr r0, r0, 0
          dout r0
          nl

          mov sp, fp       ; }
          pop fp
          pop lr
          ret
```

```
;==============================================================
                           ; class B: public A
                           ; {
                           ;    public:
                           ;       void set(int n, int m);
                           ;       void display();
                           ;    private:
                           ;       int y;
                           ; };
@B@set$ii:push lr          ; void B::set(int n, int m): public A
          push fp          ; {
          mov fp, sp

          ldr r0, fp, 3    ;    x = n;
          ldr r1, fp, 2
          str r0, r1, 0

          ldr r0, fp, 4    ;    y = m;
          ldr r1, fp, 2
          str r0, r1, 1

          mov sp, fp       ; }
          pop fp
          pop lr
          ret

@B@display$v:              ; void B::display()
          push lr          ; {
          push fp
          mov fp, sp

          ldr r0, fp, 2    ;    cout << x << " " << y << endl;
          ldr r0, r0, 0
          dout r0
          mov r0, ' '
          aout
          ldr r0, fp, 2
          ldr r0, r0, 1
          dout r0
          nl

          mov sp, fp       ; }
          pop fp
          pop lr
          ret
```

```
;==============================================================
main:       push lr          ; int main()
            push fp          ; {
            mov fp, sp

            sub sp, sp, 1    ;    A *aptr;
            sub sp, sp, 1    ;    B *bptr;

            mov r1, 1        ;    aptr = new A;
            bl malloc
            str r0, fp, -1

            mov r0, 1        ;    aptr->set(1);
            push r0
            ldr r0, fp, -1
            push r0
            bl @A@set$i
            add sp, sp, 2

            ldr r0, fp, -1   ;    aptr->display();
            push r0
            bl @A@display$v
            add sp, sp, 1

            mov r1, 2        ;    bptr = new B;
            bl malloc
            str r0, fp, -2

            mov r0, 3        ;    bptr->set(2, 3);
            push r0
            mov r0, 2
            push r0
            ldr r0, fp, -2
            push r0
            bl @B@set$ii
            add sp, sp, 3

            ldr r0, fp, -2   ;    bptr->display();
            push r0
            bl @B@display$v
            add sp, sp, 1

            ldr r0, fp, -2   ;    aptr = bptr;
            str r0, fp, -1

            ldr r0, fp, -1   ;    aptr->display();
            push r0
            bl @A@display$v
            add sp, sp, 1

            mov r0, 0        ;    return 0;
```

Calls A-level display

Calls B-level display

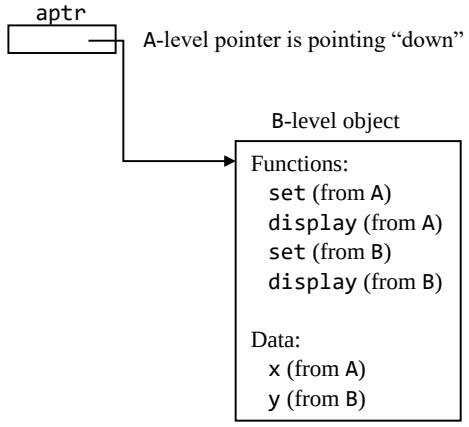
```
          mov sp, fp
          pop fp
          pop lr
          ret

malloc:   ld r0, @avail
          add r1, r0, r1
          st r1, @avail
          ret
@avail:   .word @avail+1   ; }
```

Pointer Can Point Down

```
49    aptr = bptr;
```

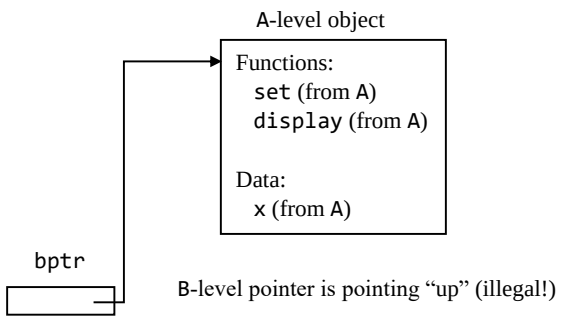
and line 129 in the assembly program that assigns `aptr` a pointer to a `B`-level object is legal. We get the following structure.



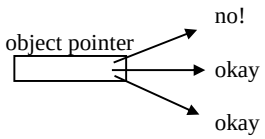
Pointer Cannot Point Up

```
bptr = aptr;
```

We are attempting to get the following structure:



Rule: An object pointer can point "across" (i.e., at an object at its own level) or "down" (i.e., to an object at a lower level) but not up (i.e., to object at a higher level).



Rule: The function called in an object depends on the level of the pointer pointing to to the object—not on the level of the object pointed to.

Virtual Functions

```
 1 // ex1602.cpp  Virtual functions
 2 #include <iostream>
 3 using namespace std;
 4 class A
 5 {
 6    public:
 7       void set(int n);
 8       virtual void display();  // display now a virtual function
 9    protected:
10       int x;
11 };
12 void A::set(int n)
13 {
14    x = n;
15 }
16 void A::display()
17 {
18    cout << x << endl;
19 }
20 //===================
21 class B: public A
22 {
23    public:
24       void set(int n, int m);
25       void display();          // this display also virtual
26    private:
27       int y;
28 };
29 void B::set(int n, int m)
30 {
31    x = n;
32    y = m;
33 }
34 void B::display()
35 {
36    cout << x << " " << y << endl;
37 }
```

```
//==============================
int main()
{
   A *aptr;
   B *bptr;
   aptr = new A;     // aptr pointing across to A-level object
   aptr->set(1);
   aptr->display();  // A-level display called
   bptr = new B;     // bptr pointing across to B-level object
   bptr->set(2, 3);
   bptr->display();  // B-level display called
   aptr = bptr;      // aptr now pointing down to B-level object
   aptr->display();  // B-level display called
   return 0;
}
```

Output With and Without virtual

Output with the `virtual` keyword:

```
1
2 3
2 3
```

(`B`-level `display` function called)

Output without the `virtual` keyword:

```
1
2 3
2
```

(`A`-level `display` function called)

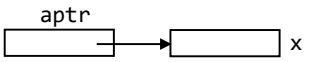
How Does This Work?

Commented [ADR1]:

```
cin > z;
if (z < 0)
    aptr = new A;   // creates A-level object. Pointer assigned to aptr.
else
    aptr = new B    // creates B-level object. Pointer assigned to aptr.
aptr->display();    // impossible to know at compile time what is in aptr.
```

```
; aptr = new A;
mov r1, 1       ; load r1 with size of allocation request
bl malloc       ; allocate storage
str r0, fp, -1  ; store pointer to allocated storage into aptr
```

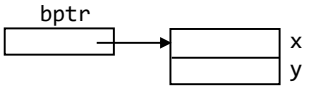
We get



If the `display` function is not virtual, then the assembly code for line 46 is

```
; bptr = new B;
mov r1, 2       ; load r1 with size of allocation request
bl malloc       ; allocate storage
str r0, fp, -2  ; store pointer to allocated storage into bptr
```

We get




```
 aptr = bptr;        // aptr is pointing down to B object
; aptr->display();
ldr r0, fp, -1
push r0
call @A@display$v   // calls A-level display function
add sp, sp, 1
```

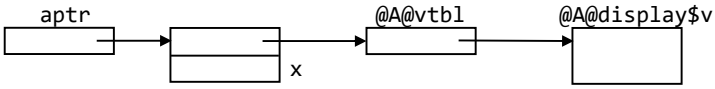
With Virtual Function

```
; aptr = new A;
mov r1, 2        ; load r1 with size of allocation request
bl malloc        ; allocate storage
str r0, fp, -1   ; store pointer to allocated storage into aptr
lea r1, @A@vtbl  ; get address of virtual table
str r1, r0, 0    ; store addr of virt tab in 1st word of object
```

where `@A@vtbl` is defined at the bottom of the program with

```
@A@vtbl:  .word @A@display$v
```

We get



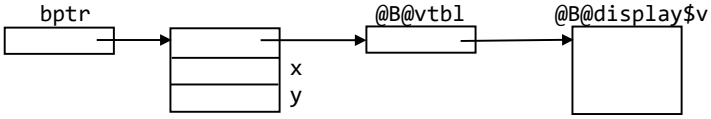
The assembly code for line 46 is

```
; bptr = new B;
mov r1, 3        ; load r1 with size of allocation request
bl malloc        ; allocate storage
str r0, fp, -2   ; store pointer to allocated storage into bptr
lea r1, @B@vtbl  ; get address of virtual table
str r1, r0, 0    ; store addr of virt tab in 1st word of object
```

where `@B@vtbl` is defined at the bottom of the program with

```
@B@vtbl:  .word @B@display$v
```

We get




```
ldr r0, fp, -1   ; get aptr
push r0          ; pass pointer in aptr to display
ldr r0, r0, 0    ; get pointer to virtual function table
ldr r0, r0, 0    ; get pointer to virtual function
blr r0           ; call virtual function
add sp, sp, 1    ; remove parameter
```

```
; ex1602.a  Virtual functions
startup:  bl main
          halt
;==============================================================
                           ; #include <iostream>
                           ; using namespace std;
                           ; class A
                           ; {
                           ;    public:
                           ;       void set(int n);
                           ;       virtual void display();
                           ;    protected:
                           ;       int x;
                           ; };
@A@set$i: push lr          ; A::set(int n)
          push fp          ; {
          mov fp, sp

          ldr r0, fp, 3    ;    x = n;
          ldr r1, fp, 2
          str r0, r1, 1

          mov sp, fp       ; }
          pop fp
          pop lr
          ret

@A@display$v:              ; void A::display()
          push lr          ; {
          push fp
          mov fp, sp

          ldr r0, fp, 2    ;    cout << x << endl;
          ldr r0, r0, 1
          dout r0
          nl

          mov sp, fp       ; }
          pop fp
          pop lr
          ret
```

```
;==============================================================
                            ; class B: public A
                            ; {
                            ;    public:
                            ;       void set(int n, int m);
                            ;       void display();
                            ;    private:
                            ;       int y;
                            ; };
@B@set$ii:push lr           ; B::set(int n, int m): public A
          push fp           ; {
          mov fp, sp

          ldr r0, fp, 3     ;    x = n;
          ldr r1, fp, 2
          str r0, r1, 1

          ldr r0, fp, 4     ;    y = b;
          ldr r1, fp, 2
          str r0, r1, 2

          mov sp, fp        ; }
          pop fp
          pop lr
          ret

@B@display$v:               ; void B::display()
          push lr           ; {
          push fp
          mov fp, sp

          ldr r0, fp, 2     ;    cout << x << " " << y << endl;
          ldr r0, r0, 1
          dout r0
          mov r0, ' '
          aout
          ldr r0, fp, 2
          ldr r0, r0, 2
          dout r0
          nl

          mov sp, fp        ; }
          pop fp
          pop lr
          ret
```

```
;==============================================================
main:     push lr          ; int main()
          push fp          ; {
          mov fp, sp

          sub sp, sp, 1    ;    A *aptr;
          sub sp, sp, 1    ;    B *bptr

          mov r1, 2        ;    aptr = new A;
          bl malloc
          str r0, fp, -1
          lea r1, @A@vtbl
          str r1, r0, 0

          mov r0, 1        ;    aptr->set(1);
          push r0
          ldr r0, fp, -1
          push r0
          bl @A@set$i
          add sp, sp, 2

          ldr r0, fp, -1   ;    aptr->display();
          push r0
          ldr r0, r0, 0
          ldr r0, r0, 0
          blr r0
          add sp, sp, 1

          mov r1, 3        ;    bptr = new B;
          bl malloc
          str r0, fp, -2
          lea r1, @B@vtbl
          str r1, r0, 0

          mov r0, 3        ;    bptr->set(2, 3);
          push r0
          mov r0, 2
          push r0
          ldr r0, fp, -2
          push r0
          bl @B@set$ii
          add sp, sp, 3

          ldr r0, fp, -2   ;    bptr->display();
          push r0
          ldr r0, r0, 0
          ldr r0, r0, 0
          blr r0
          add sp, sp, 1

          ldr r0, fp, -2   ;    aptr = bptr;
```

```
138          str r0, fp, -1
139
140          ldr r0, fp, -1   ;    aptr->display();
141          push r0
142          ldr r0, r0, 0
143          ldr r0, r0, 0
144          blr r0
145          add sp, sp, 1
146
147          mov r0, 0        ;    return 0;
148          mov sp, fp
149          pop fp
150          pop lr
151          ret
152                           ; }
153 @A@vtbl:  .word @A@display$v
154 @B@vtbl:  .word @B@display$v
155
156 malloc:   ld r0, @avail
157          add r1, r0, r1
158          st r1, @avail
159          ret
160 @avail:   .word @avail+1
```

Constructors

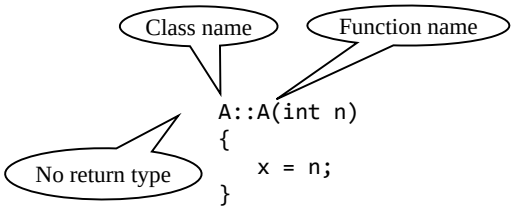
```
 1 // ex1603.cpp  Constructors
 2 #include <iostream>
 3 using namespace std;
 4 class A
 5 {
 6    public:
 7       A(int n);
 8       void display();
 9    protected:
10       int x;
11 };
12 A::A(int n)
13 {
14    x = n;
15 }
16 void A::display()
17 {
18    cout << x << endl;
19 }
20 //====================
21 class B: public A
22 {
23    public:
24       B(int n, int m);
25       void display();
26    private:
27       int y;
28 };
29 B::B(int n, int m): A(n)
30 {
31    y = m;
32 }
33 void B::display()
34 {
35    cout << x << " " << y << endl;
36 }
```

Constructor declaration

Constructor definition

```
//==============================
int main()
{
   A *aptr;
   B *bptr;
   A a(1);
   a.display();
   B b(2, 3);
   b.display();
   aptr = new A(4);
   aptr->display();
   bptr = new B(5, 6);
   bptr->display();
   return 0;
}
```

```
16 void A::display()
```

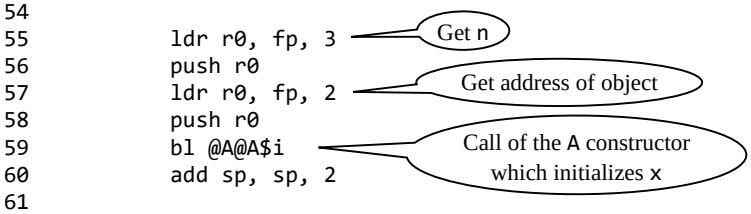


If we declare an object of type `A` with

```
42    A a(1);

46    aptr = new A(4);

29 B::B(int n, int m): A(n)
```



Output:

```
1
2 3
4
5 6
```

```
; ex1603.a  Constructors
startup:  bl main
          halt
;==============================================================
                           ; #include <iostream>
                           ; using namespace std;
                           ; class A
                           ; {
                           ;    public:
                           ;       A(int n);
                           ;       void display();
                           ;    protected:
                           ;       int x;
                           ; };
@A@A$i:   push lr          ; A::A(int n)
          push fp          ; {
          mov fp, sp

          ldr r0, fp, 3    ;    x = n;
          ldr r1, fp, 2
          str r0, r1, 0

          mov sp, fp       ; }
          pop fp
          pop lr
          ret

@A@display$v:              ; void A::display()
          push lr          ; {
          push fp
          mov fp, sp

          ldr r0, fp, 2    ;    cout << x << endl;
          ldr r0, r0, 0
          dout r0
          nl

          mov sp, fp       ; }
          pop fp
          pop lr
          ret
```

```
;==============================================================
                           ; class B: public A
                           ; {
                           ;    public:
                           ;       B(int n, int m);
                           ;       void display();
                           ;    private:
                           ;       int y;
                           ; };
@B@B$ii:  push lr          ; B::B(int n, int m): A(n)
          push fp          ; {
          mov fp, sp

          ldr r0, fp, 3
          push r0
          ldr r0, fp, 2
          push r0
          bl @A@A$i
          add sp, sp, 2

          ldr r0, fp, 4    ;    y = m;
          ldr r1, fp, 2
          str r0, r1, 1

          mov sp, fp       ; }
          pop fp
          pop lr
          ret

@B@display$v:              ; void B::display()
          push lr          ; {
          push fp
          mov fp, sp

          ldr r0, fp, 2    ;    cout << x << " " << y << endl;
          ldr r0, r0, 0
          dout r0
          mov r0, ' '
          aout
          ldr r0, fp, 2
          ldr r0, r0, 1
          dout r0
          nl

          mov sp, fp       ; }
          pop fp
          pop lr
          ret
```

```
;==============================================================
main:     push lr          ; int main()
          push fp          ; {
          mov fp, sp

          sub sp, sp, 1    ;    A *aptr;
          sub sp, sp, 1    ;    B *bptr;

          sub sp, sp, 1    ;    A a(1);
          mov r0, 1
           push r0
           add r0, fp, -3
           push r0
           bl @A@A$i
           add sp, sp, 2
           add r0, fp, -3
           push r0
           bl @A@display$v
           add sp, sp, 1

           sub sp, sp, 2    ;    B b(2, 3)
           mov r0, 3
           push r0
           mov r0, 2
           push r0
           add r0, fp, -4
           push r0
           bl @B@B$ii
           add sp, sp, 3

           add r0, fp, -4
           push r0
           bl @B@display$v
           add sp, sp, 1

           mov r1, 1        ;    aptr = new A(4);
           bl malloc
           str r0, fp, -1

           mov r0, 4
           push r0
           ldr r0, fp, -1
           push r0
           bl @A@A$i
           add sp, sp, 2

           ldr r0, fp, -1   ;    aptr->display();
           push r0
           bl @A@display$v
           add sp, sp, 1

```

```
141          mov r1, 2        ;    bptr = new B(5, 6);
142          bl malloc
143          str r0, fp, -2
144
145          mov r0, 6
146          push r0
147          mov r0, 5
148          push r0
149          ldr r0, fp, -2
150          push r0
151          bl @B@B$ii
152          add sp, sp, 3
153
154          ldr r0, fp, -2   ;    bptr->display();
155          push r0
156          bl @B@display$v
157          add sp, sp, 1
158
159          mov r0, 0        ;    return 0;
160          mov sp, fp
161          pop fp
162          pop lr
163          ret
164
165 malloc:  ld r0 @avail
166          add r1, r0, r1
167          st r1, @avail
168          ret
169 @avail:  .word @avail+1   ; }
```

Constructors Can Be Overloaded

```
B::B(): A(1)
{
   y = 2;
}
B::B(int n): A(1)
{
   y = n;
}
B::B(int n, int m): A(n)
{
   y = m;
}


B b1();          // calls first constructor
B b2(2);         // calls second constructor
B b3(2, 3);      // calls third constructor
```