**16** C++ Inheritance and Virtual Functions

Inheritance

aptr = new A;

bptr = new B;

aptr A-level object bptr B-level object

Functions:

set (from A)

display (from A)

Data:

x (from A)

Functions:

set (from A)

display (from A)

set (from B)

display (from B)

Data:

x (from A)

y (from 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 }

Indicates B is derived from A

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 }

1 ; ex1601.a Inheritance

2 startup: bl main

3 halt

4 ;==============================================================

5 ; #include <iostream>

6 ; using namespace std;

7 ; class A

8 ; {

9 ; public:

10 ; void set(int a);

11 ; void display();

12 ; protected:

13 ; int x;

14 ; };

15 @A@set$i: push lr ; A::set(int n)

16 push fp ; {

17 mov fp, sp

18

19 ldr r0, fp, 3 ; x = n;

20 ldr r1, fp, 2

21 str r0, r1, 0

22

23 mov sp, fp ; }

24 pop fp

25 pop lr

26 ret

27

28 @A@display$v: ; void A::display()

29 push lr ; {

30 push fp

31 mov fp, sp

32

33 ldr r0, fp, 2 ; cout << x << endl;

34 ldr r0, r0, 0

35 dout r0

36 nl

37

38 mov sp, fp ; }

39 pop fp

40 pop lr

41 ret

42 ;==============================================================

43 ; class B: public A

44 ; {

45 ; public:

46 ; void set(int n, int m);

47 ; void display();

48 ; private:

49 ; int y;

50 ; };

51 @B@set$ii:push lr ; void B::set(int n, int m): public A

52 push fp ; {

53 mov fp, sp

54

55 ldr r0, fp, 3 ; x = n;

56 ldr r1, fp, 2

57 str r0, r1, 0

58

59 ldr r0, fp, 4 ; y = m;

60 ldr r1, fp, 2

61 str r0, r1, 1

62

63 mov sp, fp ; }

64 pop fp

65 pop lr

66 ret

67

68 @B@display$v: ; void B::display()

69 push lr ; {

70 push fp

71 mov fp, sp

72

73 ldr r0, fp, 2 ; cout << x << " " << y << endl;

74 ldr r0, r0, 0

75 dout r0

76 mov r0, ' '

77 aout

78 ldr r0, fp, 2

79 ldr r0, r0, 1

80 dout r0

81 nl

82

83 mov sp, fp ; }

84 pop fp

85 pop lr

86 ret

87 ;==============================================================

88 main: push lr ; int main()

89 push fp ; {

90 mov fp, sp

91

92 sub sp, sp, 1 ; A \*aptr;

93 sub sp, sp, 1 ; B \*bptr;

94

95 mov r1, 1 ; aptr = new A;

96 bl malloc

97 str r0, fp, -1

98

99 mov r0, 1 ; aptr->set(1);

100 push r0

101 ldr r0, fp, -1

102 push r0

103 bl @A@set$i

104 add sp, sp, 2

105

106 ldr r0, fp, -1 ; aptr->display();

107 push r0

Calls A-level display method

108 bl @A@display$v

109 add sp, sp, 1

110

111 mov r1, 2 ; bptr = new B;

112 bl malloc

113 str r0, fp, -2

114

115 mov r0, 3 ; bptr->set(2, 3);

116 push r0

117 mov r0, 2

118 push r0

119 ldr r0, fp, -2

120 push r0

121 bl @B@set$ii

122 add sp, sp, 3

123

124 ldr r0, fp, -2 ; bptr->display();

125 push r0

Calls B-level display method

126 bl @B@display$v

127 add sp, sp, 1

128

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

130 str r0, fp, -1

131

132 ldr r0, fp, -1 ; aptr->display();

133 push r0

134 bl @A@display$v

135 add sp, sp, 1

136

137 mov r0, 0 ; return 0;

138 mov sp, fp

139 pop fp

140 pop lr

141 ret

142

143 malloc: ld r0, @avail

144 add r1, r0, r1

145 st r1, @avail

146 ret

147 @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.

aptr

A-level pointer is pointing “down”

B-level object

Functions:

set (from A)

display (from A)

set (from B)

display (from B)

Data:

x (from A)

y (from B)

Pointer Cannot Point Up

bptr = aptr;

We are attempting to get the following structure:

A-level object

Functions:

set (from A)

display (from A)

Data:

x (from A)

bptr

B-level pointer is pointing “up” (illegal!)

*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).

no!

object pointer

okay

okay

*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 }

38 //===============================

39 int main()

40 {

41 A \*aptr;

42 B \*bptr;

43 aptr = new A; // aptr pointing across to A-level object

44 aptr->set(1);

45 aptr->display(); // A-level display called

46 bptr = new B; // bptr pointing across to B-level object

47 bptr->set(2, 3);

48 bptr->display(); // B-level display called

49 aptr = bptr; // aptr now pointing down to B-level object

50 aptr->display(); // B-level display called

51 return 0;

52 }

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?

51 cin > z;

52 if (z < 0)  
53 aptr = new A; // creates A-level object. Pointer assigned to aptr.

54 else

55 aptr = new B // creates B-level object. Pointer assigned to aptr.

56 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

aptr

x

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

bptr

x

y

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

aptr @A@vtbl @A@display$v

x

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

bptr @B@vtbl @B@display$v

x

y

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

1 ; ex1602.a Virtual functions

2 startup: bl main

3 halt

4 ;==============================================================

5 ; #include <iostream>

6 ; using namespace std;

7 ; class A

8 ; {

9 ; public:

10 ; void set(int n);

11 ; virtual void display();

12 ; protected:

13 ; int x;

14 ; };

15 @A@set$i: push lr ; A::set(int n)

16 push fp ; {

17 mov fp, sp

18

19 ldr r0, fp, 3 ; x = n;

20 ldr r1, fp, 2

21 str r0, r1, 1

22

23 mov sp, fp ; }

24 pop fp

25 pop lr

26 ret

27

28 @A@display$v: ; void A::display()

29 push lr ; {

30 push fp

31 mov fp, sp

32

33 ldr r0, fp, 2 ; cout << x << endl;

34 ldr r0, r0, 1

35 dout r0

36 nl

37

38 mov sp, fp ; }

39 pop fp

40 pop lr

41 ret

42 ;==============================================================

43 ; class B: public A

44 ; {

45 ; public:

46 ; void set(int n, int m);

47 ; void display();

48 ; private:

49 ; int y;

50 ; };

51 @B@set$ii:push lr ; B::set(int n, int m): public A

52 push fp ; {

53 mov fp, sp

54

55 ldr r0, fp, 3 ; x = n;

56 ldr r1, fp, 2

57 str r0, r1, 1

58

59 ldr r0, fp, 4 ; y = b;

60 ldr r1, fp, 2

61 str r0, r1, 2

62

63 mov sp, fp ; }

64 pop fp

65 pop lr

66 ret

67

68 @B@display$v: ; void B::display()

69 push lr ; {

70 push fp

71 mov fp, sp

72

73 ldr r0, fp, 2 ; cout << x << " " << y << endl;

74 ldr r0, r0, 1

75 dout r0

76 mov r0, ' '

77 aout

78 ldr r0, fp, 2

79 ldr r0, r0, 2

80 dout r0

81 nl

82

83 mov sp, fp ; }

84 pop fp

85 pop lr

86 ret

87 ;==============================================================

88 main: push lr ; int main()

89 push fp ; {

90 mov fp, sp

91

92 sub sp, sp, 1 ; A \*aptr;

93 sub sp, sp, 1 ; B \*bptr

94

95 mov r1, 2 ; aptr = new A;

96 bl malloc

97 str r0, fp, -1

98 lea r1, @A@vtbl

99 str r1, r0, 0

100

101 mov r0, 1 ; aptr->set(1);

102 push r0

103 ldr r0, fp, -1

104 push r0

105 bl @A@set$i

106 add sp, sp, 2

107

108 ldr r0, fp, -1 ; aptr->display();

109 push r0

110 ldr r0, r0, 0

111 ldr r0, r0, 0

112 blr r0

113 add sp, sp, 1

114

115 mov r1, 3 ; bptr = new B;

116 bl malloc

117 str r0, fp, -2

118 lea r1, @B@vtbl

119 str r1, r0, 0

120

121 mov r0, 3 ; bptr->set(2, 3);

122 push r0

123 mov r0, 2

124 push r0

125 ldr r0, fp, -2

126 push r0

127 bl @B@set$ii

128 add sp, sp, 3

129

130 ldr r0, fp, -2 ; bptr->display();

131 push r0

132 ldr r0, r0, 0

133 ldr r0, r0, 0

134 blr r0

135 add sp, sp, 1

136

137 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

Constructor declaration

5 {

6 public:

7 A(int n);

8 void display();

9 protected:

10 int x;

Constructor definition

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 }

37 //===============================

38 int main()

39 {

40 A \*aptr;

41 B \*bptr;

42 A a(1);

43 a.display();

44 B b(2, 3);

45 b.display();

46 aptr = new A(4);

47 aptr->display();

48 bptr = new B(5, 6);

49 bptr->display();

50 return 0;

51 }

16 void A::display()

Class name

Function name

A::A(int n)

{

No return type

x = n;

}

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)

51 @B@B$ii: push lr ; B::B(int n, int m): A(n)

52 push fp ; {

53 mov fp, sp

54

Get n

55 ldr r0, fp, 3

56 push r0

Get address of object

57 ldr r0, fp, 2

58 push r0

Call of the A constructor which initializes x

59 bl @A@A$i

60 add sp, sp, 2

61

62 ldr r0, fp, 4 ; y = m;

63 ldr r1, fp, 2

64 str r0, r1, 1

65

66 mov sp, fp ; }

67 pop fp

68 pop lr

69 ret

Output:

1

2 3

4

5 6

1 ; ex1603.a Constructors

2 startup: bl main

3 halt

4 ;==============================================================

5 ; #include <iostream>

6 ; using namespace std;

7 ; class A

8 ; {

9 ; public:

10 ; A(int n);

11 ; void display();

12 ; protected:

13 ; int x;

14 ; };

15 @A@A$i: push lr ; A::A(int n)

16 push fp ; {

17 mov fp, sp

18

19 ldr r0, fp, 3 ; x = n;

20 ldr r1, fp, 2

21 str r0, r1, 0

22

23 mov sp, fp ; }

24 pop fp

25 pop lr

26 ret

27

28 @A@display$v: ; void A::display()

29 push lr ; {

30 push fp

31 mov fp, sp

32

33 ldr r0, fp, 2 ; cout << x << endl;

34 ldr r0, r0, 0

35 dout r0

36 nl

37

38 mov sp, fp ; }

39 pop fp

40 pop lr

41 ret

42 ;==============================================================

43 ; class B: public A

44 ; {

45 ; public:

46 ; B(int n, int m);

47 ; void display();

48 ; private:

49 ; int y;

50 ; };

51 @B@B$ii: push lr ; B::B(int n, int m): A(n)

52 push fp ; {

53 mov fp, sp

54

55 ldr r0, fp, 3

56 push r0

57 ldr r0, fp, 2

58 push r0

59 bl @A@A$i

60 add sp, sp, 2

61

62 ldr r0, fp, 4 ; y = m;

63 ldr r1, fp, 2

64 str r0, r1, 1

65

66 mov sp, fp ; }

67 pop fp

68 pop lr

69 ret

70

71 @B@display$v: ; void B::display()

72 push lr ; {

73 push fp

74 mov fp, sp

75

76 ldr r0, fp, 2 ; cout << x << " " << y << endl;

77 ldr r0, r0, 0

78 dout r0

79 mov r0, ' '

80 aout

81 ldr r0, fp, 2

82 ldr r0, r0, 1

83 dout r0

84 nl

85

86 mov sp, fp ; }

87 pop fp

88 pop lr

89 ret

90 ;==============================================================

91 main: push lr ; int main()

92 push fp ; {

93 mov fp, sp

94

95 sub sp, sp, 1 ; A \*aptr;

96 sub sp, sp, 1 ; B \*bptr;

97

98 sub sp, sp, 1 ; A a(1);

99 mov r0, 1

100 push r0

101 add r0, fp, -3

102 push r0

103 bl @A@A$i

104 add sp, sp, 2

105 add r0, fp, -3

106 push r0

107 bl @A@display$v

108 add sp, sp, 1

109

110 sub sp, sp, 2 ; B b(2, 3)

111 mov r0, 3

112 push r0

113 mov r0, 2

114 push r0

115 add r0, fp, -4

116 push r0

117 bl @B@B$ii

118 add sp, sp, 3

119

120 add r0, fp, -4

121 push r0

122 bl @B@display$v

123 add sp, sp, 1

124

125 mov r1, 1 ; aptr = new A(4);

126 bl malloc

127 str r0, fp, -1

128

129 mov r0, 4

130 push r0

131 ldr r0, fp, -1

132 push r0

133 bl @A@A$i

134 add sp, sp, 2

135

136 ldr r0, fp, -1 ; aptr->display();

137 push r0

138 bl @A@display$v

139 add sp, sp, 1

140

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