**15** C++ Objects

Introduction

A a; // creates an object named a from the class A

a

Data:

x

Functions:

set

display

a.set(5);

a.x = 10;

Object itself has a name (“a”).

Dynamically Create A Object

A \*aptr;

aptr = new A;

Object has no name

aptr

Data:

x

Functions:

set

display

aptr->set(5);

aptr->x = 20;

Structs Versus Objects

1 ; ex1501.a Structs in C++

2 startup bl main

3 halt

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

5 ; #include <iostream>

6 ; using namespace std;

7 ; struct A

8 ; {

9 ; int x;

10 ; int y;

11 ; };

13 @set$p1Aii

14 push lr ; void set(A \*r, int n, int m)

15 push fp ; {

16 mov fp, sp

17

18 ldr r0, fp, 3 ; r->x = n;

19 ldr r1, fp 2

20 str r0, r1, 0

21

22 ldr r0, fp, 4 ; r->y = m;

23 ldr r1, fp 2

24 str r0, r1, 1

25

26 mov sp, fp ; }

27 pop fp

28 pop lr

29 ret

30 ;==============================================================

31 @display$p1A ; void display(A \*r)

32 push lr ; {

33 push fp

34 mov fp, sp

35

36 ldr r0, fp, 2 ; cout << r->x << endl;

37 ldr r0, r0, 0

38 dout r0

39 nl

40

41 ldr r0, fp, 2 ; cout << r->y << endl;

42 ldr r0, r0, 1

43 dout r0

44 nl

45

46 mov sp, fp ; }

47 pop fp

48 pop lr

49 ret

50 ;==============================================================

51 main push lr ; int main()

52 push fp ; {

53 mov fp, sp

54

55 sub sp, sp, 2 ; A a, b;

56 sub sp, sp, 2

57

58 mov r0, 6 ; set(&a, 5, 6);

59 push r0

60 mov r0, 5

61 push r0

62 add r0, fp, -2

63 push r0

64 bl @set$p1Aii

65 add sp, sp, 3

66

67 add r0, fp, -2 ; display(&a);

68 push r0

69 bl @display$p1A

70 add sp, sp, 1

71

72 mov r0, 11 ; set(&b, 10, 11);

73 push r0

74 mov r0, 10

75 push r0

76 add r0, fp, -4

77 push r0

78 bl @set$p1Aii

79 add sp, sp, 3

80

81 add r0, fp, -4 ; display(&b);

82 push r0

83 bl @display$p1A

84 add sp, sp, 1

85

86 mov r0, 0 ; return 0;

87 mov sp, fp

88 pop fp

89 pop lr

90 ret

91 ; }

Equivalent C++ Program Using Objects

1 // ex1502.cpp Objects in C++

2 #include <iostream>

3 using namespace std;

4 class A

5 {

6 public:

7 void set(int n, int m);

8 void display();

9 private:

10 int x;

11 int y;

12 };

13 void A::set(int n, int m)

14 {

15 x = n;

16 y = m;

17 }

18 void A::display()

19 {

20 cout << x << endl;

21 cout << y << endl;

22 }

23 //===================

24 int main()

25 {

26 A a, b;

27 a.set(5, 6);

28 a.display();

29 b.set(10, 11);

30 b.display();

31 // b.x = 20; illegal because x is private

32 return 0;

33 }

Conceptual Picture

a b

Data: Data:

x x

y y

Functions: Functions:

set set

display display

Two questions:

1. Each object has its own set and display functions. Is this not a very inefficient use of memory—to have multiple copies of the set and display functions?
2. The set functions in the a and b objects *are identical*. How then can the set functions have a different effect? For example, the call of the set function on line 27

a.set(5, 6);

initializes the x and y *in the* a *object*, but the call of the *identical* set function on line 29

b.set(10, 11);

initializes the x and y *in the* b *object*.

Actual Picture

a b

x x Only data in objects

y y

set

One set of functions

display

1 ; ex1502.a Objects in C++

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 ; void display();

12 ; private:

13 ; int x;

14 ; int y;

15 ; };

16

17 @A@set$ii

18 push lr ; void A::set(int n, int m)

19 push fp ; {

20 mov fp, sp

21

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

23 ldr r1, fp, 2

24 str r0, r1, 0

25

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

27 ldr r1, fp, 2

28 str r0, r1, 1

29

30 mov sp, fp ; }

31 pop fp

32 pop lr

33 ret

34

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

36 push lr ; {

37 push fp

38 mov fp, sp

39

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

41 ldr r0, r0, 0

42 dout r0

43 nl

44

45 ldr r0, fp, 2 ; cout << y << endl;

46 ldr r0, r0, 1

47 dout r0

48 nl

49

50 mov sp, fp ; }

51 pop fp

52 pop lr

53 ret

54 ;==============================================================

55 main push lr ; int main()

56 push fp ; {

57 mov fp, sp

58

59 add sp, sp, -2 ; A a, b;

60 add sp, sp, -2

61

62 mov r0, 6 ; a.set(5, 6);

63 push r0

64 mov r0, 5

Get address of a

65 push r0

66 add r0, fp, -2

67 push r0

68 bl @A@set$ii

Get address of a

69 add sp, sp, 3

70

71 add r0, fp, -2 ; a.display();

72 push r0

73 bl @A@display$v

74 add sp, sp, 1

75

76 mov r0, 11 ; b.set(10, 11);

77 push r0

78 mov r0, 10

79 push r0

Get address of b

80 add r0, fp, -4

81 push r0

82 bl @A@set$ii

Get address of b

83 add sp, sp, 3

84

85 add r0, fp, -4 ; b.display();

86 push r0

87 bl @A@display$v

88 add sp, sp, 1

89

90 mov r0, 0 ; return 0;

91 mov sp, fp

92 pop fp

93 pop lr

94 ret

95 ; }

Creating Structs with malloc and Objects with new

To dynamically allocate the structs, we use

Casts the pointer returned

by malloc to A \*

p = (A \*)malloc(sizeof(A));

q = (A \*)malloc(sizeof(A));

Code for object version Struct version Object version

1 main push lr int main() int main()

2 push fp { {

3 mov fp, sp

4 sub sp, sp, 1 A \*p, \*q; A \*p, \*q;

5 sub sp, sp, 1

6

7 mov r1, 2 p = (A \*)malloc(sizeof(A)); p = new A;

9 bl malloc

11 str r0, fp, -1

12

13 mov r1, 2 q = (A \*)malloc(sizeof(A)); q = new A;

15 bl malloc

17 str r0, fp, -2

19

20 mov r0, 6 set(p, 5, 6); p->set(5, 6);

21 push r0

22 mov r0, 5

23 push r0

24 ldr r0, fp, -1

25 push r0

26 bl @A@set$ii

27 add sp, sp, 3

28

29 ldr r0, fp, -1 display(p) p->display();

30 push r0

31 bl @A@display$v

32 add sp, sp, 1

33

34 mov r0, 11 set(q, 10, 11); q->set(10, 11);

35 push r0

36 mov r0, 10

37 push r0

38 ldr r0, fp, -2

39 push r0

40 bl @A@set$ii

41 add sp, sp, 3

42

43 ldr r0, fp, -2 display(q); q->display();

44 push r0

45 bl @A@display$v

46 add sp, sp, 1

47

48 mov r0, 0 return 0; return 0;

49 mov sp, fp

50 pop fp

51 pop lr

52 ret

53 } }

54 malloc ld r0, @avail

55 add r1, r0, r1

56 st r1, @avail

57 ret

58 @avail .word @avail+1