

# Contents

<b>1</b>	<b>Basics of C++ Programming</b>	<b>4</b>
1.1	Namespace . . . . .	4
1.2	Manipulators . . . . .	4
1.3	Dynamic memory allocation . . . . .	5
1.4	Functions . . . . .	5
1.4.1	Default Arguments . . . . .	5
1.4.2	Inline Functions . . . . .	6
1.4.3	Function overloading – Different types of arguments . . . . .	7
1.4.4	Function overloading – Different number of arguments . . . . .	7
1.4.5	Pass by value . . . . .	8
1.4.6	Pass by reference . . . . .	8
1.4.7	Pass by pointer . . . . .	9
1.4.8	Return by value . . . . .	10
1.4.9	Return by reference . . . . .	10
1.4.10	Return by pointer . . . . .	11
<b>2</b>	<b>Classes and Objects</b>	<b>13</b>
2.1	Constructors . . . . .	13
2.1.1	Default Constructor . . . . .	13
2.1.2	Parameterized Constructor . . . . .	13
2.1.3	Copy constructor . . . . .	14
2.1.4	Default copy constructor . . . . .	15
2.2	Destructor . . . . .	16
2.3	Objects as Function Arguments . . . . .	16
2.3.1	Pass by value . . . . .	16
2.3.2	Pass by reference . . . . .	18
2.3.3	Pass by pointer . . . . .	20

2.4	Structure . . . . .	21
2.4.1	Showing all the extensions to structure . . . . .	21
2.5	Static data members in class . . . . .	21
2.6	Member functions defined outside the class . . . . .	22
<b>3</b>	<b>Operator Overloading</b>	<b>24</b>
3.1	Overloading unary operator . . . . .	24
3.2	Overloading binary operator . . . . .	25
3.3	Overloading binary operator with friend function . . . . .	28
3.4	Data Conversion . . . . .	29
3.4.1	From basic to basic . . . . .	29
3.4.2	From basic to user defined . . . . .	30
3.4.3	From user defined to basic . . . . .	30
3.4.4	From user defined to user defined . . . . .	31
<b>4</b>	<b>Inheritance</b>	<b>37</b>
4.1	Overriding member functions . . . . .	37
4.2	Types of inheritance . . . . .	38
4.2.1	Single . . . . .	38
4.2.2	Multiple . . . . .	38
4.2.3	Hierarchical . . . . .	39
4.2.4	Multilevel . . . . .	40
4.2.5	Hybrid . . . . .	41
4.3	Ambiguity in multiple inheritance . . . . .	41
4.4	Ambiguity in multipath inheritance . . . . .	41
4.5	Aggregation . . . . .	42
<b>5</b>	<b>Virtual function, Polymorphism and Miscellaneous C++ Features</b>	<b>45</b>
5.1	Virtual function . . . . .	45
5.2	Pure virtual function . . . . .	47
5.3	Virtual destructor . . . . .	48
5.4	Friend class . . . . .	49
5.5	Friend Function . . . . .	50
5.6	Static function . . . . .	51
<b>6</b>	<b>Function Templates and Exception Handling</b>	<b>52</b>

6.1	Function templates . . . . .	52
6.2	Function templates with multiple arguments . . . . .	52
6.3	Class template . . . . .	53
6.4	Inheritance in template class . . . . .	54
6.5	Exception Handling . . . . .	55
6.6	Multiple catch blocks for a single try block . . . . .	55
6.7	Catch all exceptions . . . . .	56
<b>7</b>	<b>File handling and Streams</b>	<b>58</b>
7.1	Read contents of a file character by character . . . . .	58
7.2	Writing as well as reading from the file . . . . .	58
7.3	Write and Read to/from a single object . . . . .	59
7.4	Write multiple objects . . . . .	60
7.5	Read multiple objects . . . . .	61
7.6	Overloading insertion operator . . . . .	62
7.7	Overloading extraction operator . . . . .	63
7.8	Overloading both insertion and extraction operator . . . . .	64
<b>8</b>	<b>Extra</b>	<b>66</b>
8.1	Reverse string . . . . .	66
8.2	Stack implementation using templates . . . . .	67
8.3	2078 - 1 . . . . .	68
8.4	2076 - 1 . . . . .	69

# Chapter 1

## Basics of C++ Programming

### 1.1 Namespace

```
1  #include <iostream>
2
3  using namespace std;
4
5  namespace foo {
6  int bar = 10;
7  }
8
9  int
10 main(void)
11 {
12     cout << foo::bar << endl;
13
14     return 0;
15 }
```

### 1.2 Manipulators

```
1  #include <iostream>
2  #include <iomanip>
3
4  using namespace std;
5
6  int
7  main(void)
8  {
9      float foo;
10     string bar;
11 }
```

```

12     foo = 123.456;
13     bar = "buzz";
14
15     cout << setprecision(4) << foo << endl;
16     cout << setw(10) << bar << endl;
17
18     return 0;
19 }
20
21 /*
22 Output:
23
24 123.5
25      buzz
26 */

```

## 1.3 Dynamic memory allocation

```

1  #include <iostream>
2
3  using namespace std;
4
5  int
6  main(void)
7  {
8      int *foo;
9
10     foo = new int;
11     *foo = 5;
12
13     cout << *foo << endl;
14
15     delete foo;
16
17     return 0;
18 }

```

## 1.4 Functions

### 1.4.1 Default Arguments

- If value for the parameter is not passed when a function is called, the default value is used, but if a value is specified this default value is ignored and the passed value is used instead.

```

1  #include <iostream>

```

```

2
3  using namespace std;
4
5  int
6  add(int num1, int num2 = 1)
7  {
8      return num1 + num2;
9  }
10
11 int
12 main(void)
13 {
14     int num;
15
16     num = 10;
17
18     cout << add(num, 5) << endl;    /* NOT using the default parameter */
19     cout << add(num) << endl;      /* Using the default parameter */
20
21     return 0;
22 }

```

## 1.4.2 Inline Functions

- A function which is expanded inline by the compiler each time its call is appeared instead of jumping to the called function as usual is called inline function.

```

1  #include <iostream>
2
3  using namespace std;
4
5  inline void
6  print_hello(void)
7  {
8      cout << "Hello, World!" << endl;
9  }
10
11 int
12 main(void)
13 {
14     print_hello();
15
16     return 0;
17 }

```

### 1.4.3 Function overloading – Different types of arguments

```
1  #include <iostream>
2
3  using namespace std;
4
5  int
6  add(int num1, int num2)
7  {
8      return num1 + num2;
9  }
10
11 float
12 add(float num1, float num2)
13 {
14     return num1 + num2;
15 }
16
17 int
18 main(void)
19 {
20     cout << add(5, 2) << endl;
21     cout << add((float)5.5, (float)3.2) << endl;
22
23     return 0;
24 }
```

### 1.4.4 Function overloading – Different number of arguments

```
1  #include <iostream>
2
3  using namespace std;
4
5  int
6  add(int num1, int num2)
7  {
8      return num1 + num2;
9  }
10
11 int
12 add(int num1, int num2, int num3)
13 {
14     return num1 + num2 + num3;
15 }
16
17 int
18 main(void)
```

```

19 {
20     cout << add(4, 2) << endl;
21     cout << add(6, 4, 2) << endl;
22
23     return 0;
24 }

```

### 1.4.5 Pass by value

```

1  #include <iostream>
2
3  using namespace std;
4
5  void
6  exchange(int num1, int num2)
7  {
8      int temp_num;
9
10     temp_num = num1;
11     num1 = num2;
12     num2 = temp_num;
13 }
14
15 int
16 main(void)
17 {
18     int num1, num2;
19
20     num1 = 5;
21     num2 = 7;
22
23     cout << "Before exchange: " << num1 << "-" << num2 << endl;
24
25     exchange(num1, num2);
26
27     cout << "After exchange: " << num1 << "-" << num2 << endl;
28
29     return 0;
30 }

```

### 1.4.6 Pass by reference

```

1  #include <iostream>
2
3  using namespace std;
4

```



```

5  void
6  exchange(int &num1, int &num2)
7  {
8      int temp_num;
9
10     temp_num = num1;
11     num1 = num2;
12     num2 = temp_num;
13 }
14
15 int
16 main(void)
17 {
18     int num1, num2;
19
20     num1 = 5;
21     num2 = 7;
22
23     cout << "Before exchange: " << num1 << "-" << num2 << endl;
24
25     exchange(num1, num2);
26
27     cout << "After exchange: " << num1 << "-" << num2 << endl;
28
29     return 0;
30 }

```

### 1.4.7 Pass by pointer

```

1  #include <iostream>
2
3  using namespace std;
4
5  void
6  exchange(int *num1, int *num2)
7  {
8      int temp_num;
9
10     temp_num = *num1;
11     *num1 = *num2;
12     *num2 = temp_num;
13 }
14
15 int
16 main(void)
17 {
18     int num1, num2;

```

```

19
20     num1 = 5;
21     num2 = 7;
22
23     cout << "Before exchange: " << num1 << "-" << num2 << endl;
24
25     exchange(&num1, &num2);
26
27     cout << "After exchange: " << num1 << "-" << num2 << endl;
28
29     return 0;
30 }

```

### 1.4.8 Return by value

```

1  #include <iostream>
2
3  using namespace std;
4
5  int
6  largest(int num1, int num2)
7  {
8      return num1 > num2 ? num1 : num2;
9  }
10
11 int
12 main(void)
13 {
14     int num1, num2, largest_num;
15
16     num1 = 5;
17     num2 = 7;
18
19     largest_num = largest(num1, num2);
20     cout << largest_num << endl;
21
22     return 0;
23 }

```

### 1.4.9 Return by reference

```

1  #include <iostream>
2
3  using namespace std;
4
5  int &

```

```

6  largest(int &num1, int &num2)
7  {
8      return num1 > num2 ? num1 : num2;
9  }
10
11  int
12  main(void)
13  {
14      int num1, num2;
15
16      num1 = 5;
17      num2 = 7;
18
19      cout << num1 << "-" << num2 << endl;
20
21      largest(num1, num2) = 0;
22
23      cout << num1 << "-" << num2 << endl;
24
25      return 0;
26  }

```

### 1.4.10 Return by pointer

```

1  #include <iostream>
2
3  using namespace std;
4
5  int *
6  largest(int *num1, int *num2)
7  {
8      return *num1 > *num2 ? num1 : num2;
9  }
10
11  int
12  main(void)
13  {
14      int num1, num2;
15
16      num1 = 5;
17      num2 = 7;
18
19      cout << num1 << "-" << num2 << endl;
20
21      *(largest(&num1, &num2)) = 0;
22
23      cout << num1 << "-" << num2 << endl;

```

```
24  
25     return 0;  
26 }
```

# Chapter 2

## Classes and Objects

### 2.1 Constructors

#### 2.1.1 Default Constructor

```
1  #include <iostream>
2
3  using namespace std;
4
5  class foo {
6  public:
7      foo()
8      {
9          cout << "Foo object created" << endl;
10     }
11 };
12
13 int
14 main(void)
15 {
16     foo f;
17
18     return 0;
19 }
```

#### 2.1.2 Parameterized Constructor

```
1  #include <iostream>
2
3  using namespace std;
4
5  class foo {
6  private:
```

```

7         int bar;
8     public:
9         foo()
10        {
11            cout << "No parameter was given" << endl;
12        }
13
14        foo(int b)
15        {
16            cout << "Bar was given" << endl;
17            bar = b;
18        }
19    };
20
21    int
22    main(void)
23    {
24        foo f1, f2(5);
25
26        return 0;
27    }

```

### 2.1.3 Copy constructor

```

1  #include <iostream>
2
3  using namespace std;
4
5  class foo {
6  private:
7      int bar;
8  public:
9      foo(int b)
10     {
11         cout << "A value to bar is given" << endl;
12         bar = b;
13     }
14
15     foo(foo &f)
16     {
17         cout << "A foo object instance is given" << endl;
18         bar = f.bar;
19     }
20 };
21
22 int
23 main(void)

```

```

24 {
25     foo f1(5), f2(f1);
26
27     return 0;
28 }

```

## 2.1.4 Default copy constructor

```

1  #include <iostream>
2
3  using namespace std;
4
5  class foo {
6  private:
7      int bar;
8  public:
9      foo(int b)
10     {
11         cout << "A value to bar is given" << endl;
12         bar = b;
13     }
14
15     void
16     print_bar(void)
17     {
18         cout << bar << endl;
19     }
20 };
21
22 int
23 main(void)
24 {
25     foo f1(5), f2(f1);
26
27     f1.print_bar();
28     f2.print_bar();
29
30     return 0;
31 }
32
33 /*
34 Output:
35
36 A value to bar is given
37 5
38 5
39 */

```

## 2.2 Destructor

```
1  #include <iostream>
2
3  using namespace std;
4
5  class foo {
6  public:
7      foo()
8      {
9          cout << "Foo object is created" << endl;
10     }
11
12     ~foo()
13     {
14         cout << "Foo object is destroyed" << endl;
15     }
16 };
17
18 int
19 main(void)
20 {
21     foo f;
22
23     return 0;
24 }
```

## 2.3 Objects as Function Arguments

### 2.3.1 Pass by value

```
1  #include <iostream>
2
3  using namespace std;
4
5  class distances {
6  private:
7      int feet;
8      int inches;
9
10     void
11     recalc(void)
12     {
13         feet = feet + inches/12;
14         inches = inches % 12;
```



```

15     }
16 public:
17     distances(void)
18     {
19         feet = 0;
20         inches = 0;
21     }
22
23     void
24     set_distance(int f, int i)
25     {
26         feet = f;
27         inches = i;
28
29         recalc();
30     }
31
32     void
33     add_distance(distances d)
34     {
35         feet += d.feet;
36         inches += d.inches;
37
38         recalc();
39     }
40
41     void
42     print_distances(void)
43     {
44         cout << "Feet: " << feet << ", Inches: " << inches << endl;
45     }
46 };
47
48 int
49 main(void)
50 {
51     distances d1, d2;
52
53     d1.set_distance(4, 10);
54     d2.set_distance(2, 4);
55     d2.add_distance(d1);
56
57     d1.print_distances();
58     d2.print_distances();
59
60     return 0;
61 }
62

```

```
63  /*
64  Output:
65
66  Feet: 4, Inches: 10
67  Feet: 7, Inches: 2
68  */
```

### **2.3.2 Pass by reference**

```

1  #include <iostream>
2
3  using namespace std;
4
5  class foo {
6  private:
7      int bar;
8  public:
9      void
10     set_bar(int b)
11     {
12         bar = b;
13     }
14
15     /* This function changes the value of `bar` in the passed `foo` object
16      * with the value of `bar` in the current object. */
17     void
18     change_bar(foo &f)
19     {
20         f.bar = bar;
21     }
22
23     void
24     print_bar(void)
25     {
26         cout << bar << endl;
27     }
28 };
29
30 int
31 main(void)
32 {
33     foo b1, b2;
34
35     b1.set_bar(5);
36     b1.change_bar(b2);
37
38     b1.print_bar();
39     b2.print_bar();
40
41     return 0;
42 }
43
44 /*
45 Output:
46 5
47 5
48 */

```

### 2.3.3 Pass by pointer

```
1  #include <iostream>
2
3  using namespace std;
4
5  class foo {
6  private:
7      int bar;
8  public:
9      void
10     set_bar(int b)
11     {
12         bar = b;
13     }
14
15     /* This function changes the value of `bar` in the passed `foo` object
16      * with the value of `bar` in the current object. */
17     void
18     change_bar(foo *f)
19     {
20         f->bar = bar;
21     }
22
23     void
24     print_bar(void)
25     {
26         cout << bar << endl;
27     }
28 };
29
30 int
31 main(void)
32 {
33     foo b1, b2;
34
35     b1.set_bar(5);
36     b1.change_bar(&b2);
37
38     b1.print_bar();
39     b2.print_bar();
40
41     return 0;
42 }
43
44 /*
45 Output:
46 5
```

```
47 5
48 */
```

## 2.4 Structure

### 2.4.1 Showing all the extensions to structure

```
1  #include <iostream>
2
3  using namespace std;
4
5  struct foo {
6  private:
7      int bar;
8  public:
9      void
10     set_bar(int b)
11     {
12         bar = b;
13     }
14
15     void
16     print_bar(void)
17     {
18         cout << bar << endl;
19     }
20 };
21
22 int
23 main(void)
24 {
25     foo f;
26
27     f.set_bar(5);
28     f.print_bar();
29
30     return 0;
31 }
```

## 2.5 Static data members in class

```
1  #include <iostream>
2  using namespace std;
3
```

```

4  class foo {
5  private:
6      static int foo_count;
7  public:
8
9      foo()
10     {
11         foo_count++;
12     }
13
14     void
15     print_count(void)
16     {
17         cout << foo_count << endl;
18     }
19 };
20
21 int foo::foo_count = 0;
22
23 int
24 main(void)
25 {
26     foo f1, f2;
27
28     f1.print_count();
29
30     return 0;
31 }

```

## 2.6 Member functions defined outside the class

```

1  #include <iostream>
2  using namespace std;
3
4  class foo {
5  private:
6      int bar;
7  public:
8      void
9      set_bar(int f);
10
11      void
12      print_bar(void);
13 };
14
15 void
16 foo::set_bar(int f)
17 {
18     bar = f;
19 }
20
21 void
22 foo::print_bar(void)
23 {
24     cout << bar << endl;
25 }
26
27 int
28 main(void)
29 {
30     foo f;
31
32     f.set_bar(5);
33     f.print_bar();
34
35     return 0;
36 }

```

# Chapter 3

## Operator Overloading

### 3.1 Overloading unary operator

#### Prefix and Postfix

```
1  #include <iostream>
2  using namespace std;
3
4  class rectangle {
5  private:
6      int length;
7      int breadth;
8
9  public:
10     void
11     set_data(int l, int b)
12     {
13         length = l;
14         breadth = b;
15     }
16
17     void
18     print_data(void)
19     {
20         cout << "Length: " << length << ", Breadth: " << breadth
21             << endl;
22     }
23
24     /* pre-increment */
25     void
26     operator ++(void)
27     {
28         ++length;
29         ++breadth;
```



```

30     }
31
32     /* post-increment */
33     void
34     operator ++(int)
35     {
36         length++;
37         breadth++;
38     }
39 };
40
41 int
42 main(void)
43 {
44     rectangle r1, r2;
45
46     r1.set_data(5, 6);
47     r2.set_data(7, 3);
48
49     r1++;
50     ++r2;
51
52     r1.print_data();
53     r2.print_data();
54
55     return 0;
56 }

```

## 3.2 Overloading binary operator

### Plus

```

1  #include <iostream>
2  using namespace std;
3
4  class rectangle {
5  private:
6      int length;
7      int breadth;
8
9  public:
10     void
11     set_data(int l, int b)
12     {
13         length = l;
14         breadth = b;

```

```

15     }
16
17     void
18     print_data(void)
19     {
20         cout << "Length: " << length << ", Breadth: " << breadth
21             << endl;
22     }
23
24     /* plus */
25     rectangle
26     operator +(rectangle r)
27     {
28         rectangle temp_r;
29
30         temp_r.length = length + r.length;
31         temp_r.breadth = breadth + r.breadth;
32
33         return temp_r;
34     }
35 };
36
37 int
38 main(void)
39 {
40     rectangle r1, r2, r3;
41
42     r1.set_data(5, 6);
43     r2.set_data(7, 3);
44
45     r3 = r1 + r2;
46     r3.print_data();
47
48     return 0;
49 }

```

## Comparison

```

1  #include <iostream>
2  using namespace std;
3
4  class length {
5  private:
6      int data;
7  public:
8      void
9      set_data(int d)

```

```

10     {
11         data = d;
12     }
13
14     void
15     print_data(void)
16     {
17         cout << "Length = " << data << endl;
18     }
19
20     int
21     operator >(length l)
22     {
23         return l.data > data;
24     }
25 };
26
27 int
28 main(void)
29 {
30     length r1, r2;
31
32     cout << (r1 > r2 ? "r1 is greater" : "r2 is greater") << endl;
33
34     return 0;
35 }

```

## Assignment

```

1  #include <iostream>
2  using namespace std;
3
4  class length {
5  private:
6      int data;
7  public:
8      void
9      set_data(int d)
10     {
11         data = d;
12     }
13
14     void
15     print_data(void)
16     {
17         cout << data << endl;
18     }

```

```

19
20     void
21     operator =(length &l)
22     {
23         data = l.data - 1;
24     }
25 };
26
27 int
28 main(void)
29 {
30     length l1, l2;
31
32     l1.set_data(5);
33     l2 = l1;
34
35     l1.print_data();
36     l2.print_data();
37 }

```

### 3.3 Overloading binary operator with friend function

```

1  #include <iostream>
2  using namespace std;
3
4  class rectangle {
5  private:
6      int length;
7      int breadth;
8  public:
9      void
10     set_data(int l, int b)
11     {
12         length = l;
13         breadth = b;
14     }
15
16     void
17     print_data(void)
18     {
19         cout << "Length: " << length << ", Breadth: " << breadth << endl;
20     }
21
22     friend rectangle
23     operator +(rectangle r1, rectangle r2);
24 };

```

```

25
26 rectangle
27 operator +(rectangle r1, rectangle r2)
28 {
29     rectangle temp_r;
30
31     temp_r.length = r1.length + r2.length;
32     temp_r.breadth = r1.breadth + r2.breadth;
33
34     return temp_r;
35 }
36
37 int
38 main(void)
39 {
40     rectangle r1, r2, r3;
41
42     r1.set_data(5, 6);
43     r2.set_data(2, 5);
44
45     r3 = r1 + r2;
46     r3.print_data();
47
48     return 0;
49 }

```

## 3.4 Data Conversion

### 3.4.1 From basic to basic

```

1  #include <iostream>
2  using namespace std;
3
4  int
5  main(void)
6  {
7      float f_num;
8      int i_num;
9
10     f_num = 1234.567;
11     i_num = (int)f_num;
12
13     cout << i_num << endl;
14
15     return 0;
16 }

```

### 3.4.2 From basic to user defined

```
1  #include <iostream>
2  using namespace std;
3
4  class distances {
5  private:
6      int feet;
7      int inches;
8  public:
9      distances(int i)
10     {
11         feet = i / 12;
12         inches = i % 12;
13     }
14
15     void
16     print_data(void)
17     {
18         cout << "Feet = " << feet << ", Inches = " << inches << endl;
19     }
20 };
21
22 int
23 main(void)
24 {
25     distances d(14);
26
27     d.print_data();
28
29     return 0;
30 }
```

### 3.4.3 From user defined to basic

```
1  #include <iostream>
2  using namespace std;
3
4  class height {
5  private:
6      int feet;
7      int inches;
8  public:
9      height(int f, int i)
10     {
11         feet = f;
12         inches = i;
```

```

13     }
14
15     operator int(void)
16     {
17         int temp_i;
18
19         temp_i = (feet * 12) + inches;
20
21         return temp_i;
22     }
23 };
24
25 int
26 main(void)
27 {
28     height h(3, 7);
29     int inches;
30
31     inches = (int)h;
32
33     cout << inches << endl;
34
35     return 0;
36 }

```

### 3.4.4 From user defined to user defined

#### Routine in the source class

```

1  #include <iostream>
2  using namespace std;
3
4  class length_centimeter {
5  private:
6      int centimeter;
7  public:
8      void
9      set_centimeter(int c)
10     {
11         centimeter = c;
12     }
13
14     void
15     print_centimeter(void)
16     {
17         cout << centimeter << endl;
18     }

```

```

19 };
20
21 class length_meter {
22 private:
23     int meter;
24 public:
25     void
26     set_meter(int m)
27     {
28         meter = m;
29     }
30
31     operator length_centimeter(void)
32     {
33         length_centimeter temp_c;
34
35         temp_c.set_centimeter(meter * 100);
36
37         return temp_c;
38     }
39 };
40
41 int
42 main(void)
43 {
44     length_centimeter cm;
45     length_meter m;
46
47     m.set_meter(3);
48     cm = (length_centimeter)m;
49
50     cm.print_centimeter();
51
52     return 0;
53 }

```

### Routine in destination class

```

1  #include <iostream>
2  using namespace std;
3
4  class length_meter {
5  private:
6      int meter;
7  public:
8      void
9      set_meter(int m)

```



```

10     {
11         meter = m;
12     }
13
14     int
15     get_meter(void)
16     {
17         return meter;
18     }
19 };
20
21 class length_centimeter {
22 private:
23     int centimeter;
24 public:
25     length_centimeter(length_meter m)
26     {
27         centimeter = m.get_meter() * 100;
28     }
29
30     void
31     print_centimeter(void)
32     {
33         cout << centimeter << endl;
34     }
35 };
36
37 int
38 main(void)
39 {
40     length_meter m;
41
42     m.set_meter(3);
43
44     length_centimeter cm(m);
45     cm.print_centimeter();
46
47     return 0;
48 }

```

## ALTERNATIVE CODE

```

1  #include <iostream>
2  using namespace std;
3
4  class length_meter {
5  private:

```

```

6         int meter;
7     public:
8         void
9         set_meter(int m)
10        {
11            meter = m;
12        }
13
14        int
15        get_meter(void)
16        {
17            return meter;
18        }
19    };
20
21    class length_centimeter {
22    private:
23        int centimeter;
24    public:
25        length_centimeter(void)
26        {
27        }
28
29        length_centimeter(length_meter m)
30        {
31            centimeter = m.get_meter() * 100;
32        }
33
34        void
35        print_centimeter(void)
36        {
37            cout << centimeter << endl;
38        }
39    };
40
41    int
42    main(void)
43    {
44        length_meter m;
45        length_centimeter cm;
46
47        m.set_meter(3);
48        cm = m;
49
50        cm.print_centimeter();
51
52        return 0;
53    }

```

## Both in one

```
1  #include <iostream>
2  using namespace std;
3
4  class foo {
5  public:
6      int data;
7  };
8
9  class bar {
10 public:
11     int data;
12
13     bar(void)
14     {}
15
16     /* routine in source */
17     operator
18     foo(void)
19     {
20         foo temp_f;
21
22         temp_f.data = data;
23
24         return temp_f;
25     }
26
27     /* routine in destination */
28     bar(foo f)
29     {
30         data = f.data;
31     }
32 };
33
34 int
35 main(void)
36 {
37     foo f;
38     bar b;
39
40     /* routine in source */
41     b.data = 5;
42     f = (foo)b;
43     cout << f.data << endl;
44
45     /* routine in destination */
46     f.data = 6;
```

```
47     b = (bar)f;  
48     cout << b.data << endl;  
49  
50     return 0;  
51 }
```

# Chapter 4

## Inheritance

### 4.1 Overriding member functions

```
1  #include <iostream>
2  using namespace std;
3
4  class foo {
5  public:
6      void
7      say_hello(void)
8      {
9          cout << "Hello from foo" << endl;
10     }
11 };
12
13 class bar: public foo {
14 public:
15     void
16     say_hello(void)
17     {
18         cout << "Hello from bar" << endl;
19     }
20 };
21
22 int
23 main(void)
24 {
25     bar b;
26
27     b.foo::say_hello();
28
29     return 0;
30 }
```

## 4.2 Types of inheritance

### 4.2.1 Single

```
1  #include <iostream>
2  using namespace std;
3
4  class foo {
5  public:
6      foo(void)
7      {
8          cout << "Class foo initialized" << endl;
9      }
10 };
11
12 class bar: public foo {
13 public:
14     bar(void)
15     {
16         cout << "Class bar initialized" << endl;
17     }
18 };
19
20 int
21 main(void)
22 {
23     bar b;
24
25     return 0;
26 }
```

### 4.2.2 Multiple

```
1  #include <iostream>
2  using namespace std;
3
4  class foo {
5  public:
6      foo(void)
7      {
8          cout << "Class foo initialized" << endl;
9      }
10 };
11
12 class bar {
13 public:
```

```

14         bar(void)
15     {
16         cout << "Class bar initialized" << endl;
17     }
18 };
19
20 class buzz: public foo, public bar {
21 public:
22     buzz(void)
23     {
24         cout << "Class buzz initialized" << endl;
25     }
26 };
27
28 int
29 main(void)
30 {
31     buzz b;
32
33     return 0;
34 }

```

### 4.2.3 Hierarchical

```

1  #include <iostream>
2  using namespace std;
3
4  class vehicle {
5  public:
6      vehicle(void)
7      {
8          cout << "Vehicle initialized" << endl;
9      }
10 };
11
12 class bike: public vehicle {
13 public:
14     bike(void)
15     {
16         cout << "Bike initialized" << endl;
17     }
18 };
19
20 class car: public vehicle {
21 public:
22     car(void)
23     {

```

```

24         cout << "Car initialized" << endl;
25     }
26 };
27
28 int
29 main(void)
30 {
31     bike b;
32     car c;
33
34     return 0;
35 }

```

#### 4.2.4 Multilevel

```

1  #include <iostream>
2  using namespace std;
3
4  class vehicle {
5  public:
6      vehicle(void)
7      {
8          cout << "Vehicle initialized" << endl;
9      }
10 };
11
12 class bike: public vehicle {
13 public:
14     bike(void)
15     {
16         cout << "Bike initialized" << endl;
17     }
18 };
19
20 class xpulse: public bike {
21 public:
22     xpulse(void)
23     {
24         cout << "Xpulse initialized" << endl;
25     }
26 };
27
28 int
29 main(void)
30 {
31     xpulse x;
32

```



```

33         return 0;
34     }

```

## 4.2.5 Hybrid

Jhew lagyo

## 4.3 Ambiguity in multiple inheritance

```

1  #include <iostream>
2  using namespace std;
3
4  class foo {
5  public:
6      int data;
7  };
8
9  class bar {
10 public:
11     int data;
12 };
13
14 class buzz: public foo, public bar {
15 };
16
17 int
18 main(void)
19 {
20     buzz b;
21
22     /* b.data = 5; */
23     b.foo::data = 2;
24     b.bar::data = 3;
25
26     return 0;
27 }

```

## 4.4 Ambiguity in multipath inheritance

```

1  #include <iostream>
2  using namespace std;
3
4  class A {
5  public:

```

```

6         int a;
7     };
8
9     class B: virtual public A {
10    public:
11         int b;
12    };
13
14    class C: virtual public A {
15    public:
16         int c;
17    };
18
19    class D: public B, public C {
20    public:
21         int d;
22    };
23
24    int
25    main(void)
26    {
27        D o;
28
29        o.a = 1;
30        o.b = 2;
31        o.c = 3;
32        o.d = 4;
33
34        cout << o.a << endl;
35        cout << o.b << endl;
36        cout << o.c << endl;
37        cout << o.d << endl;
38    }

```

## 4.5 Aggregation

```

1  #include <iostream>
2  using namespace std;
3
4  class employee {
5  private:
6      string e_id;
7      string e_name;
8  public:
9      void
10     set_data(string id, string name)

```

```

11     {
12         e_id = id;
13         e_name = name;
14     }
15
16     string
17     get_id(void)
18     {
19         return e_id;
20     }
21
22     string
23     get_name(void)
24     {
25         return e_name;
26     }
27 };
28
29 class college {
30 private:
31     employee e;
32
33     string c_id;
34     string c_name;
35 public:
36     void
37     set_data(string id, string name)
38     {
39         c_id = id;
40         c_name = name;
41     }
42
43     void
44     set_employee_data(string id, string name)
45     {
46         e.set_data(id, name);
47     }
48
49     void
50     print_data(void)
51     {
52         cout << "College ID: " << c_id << ", Name: " << c_name << endl;
53         cout << "Employee ID: " << e.get_id() << ", Name: " <<
↵ e.get_name() << endl;
54     }
55 };
56
57 int

```

```
58  main(void)
59  {
60      college c;
61
62      c.set_data("12", "foo");
63      c.set_employee_data("34", "bar");
64
65      c.print_data();
66
67      return 0;
68  }
```

# Chapter 5

## Virtual function, Polymorphism and Miscellaneous C++ Features

### 5.1 Virtual function

```
1  #include <iostream>
2  using namespace std;
3
4  class Player {
5  public:
6      virtual void
7      say_hello(void)
8      {
9          cout << "Hello from Player" << endl;
10     }
11 };
12
13 class Ram: public Player {
14 public:
15     void
16     say_hello(void)
17     {
18         cout << "Hello from Ram" << endl;
19     }
20 };
21
22 class Shyam: public Player {
23 public:
24     void
25     say_hello(void)
26     {
27         cout << "Hello from Shyam" << endl;
28     }
29 };
```

```

30
31 class Hari: public Player {
32 public:
33     void
34     say_hello(void)
35     {
36         cout << "Hello from Hari" << endl;
37     }
38 };
39
40 class Sita: public Player {
41 public:
42     void
43     say_hello(void)
44     {
45         cout << "Hello from Sita" << endl;
46     }
47 };
48
49 int
50 main(void)
51 {
52     Player *p;
53
54     Ram r;
55     Shyam s;
56     Hari h;
57     Sita si;
58
59     p = &r;
60     p->say_hello();
61
62     p = &s;
63     p->say_hello();
64
65     p = &h;
66     p->say_hello();
67
68     p = &si;
69     p->say_hello();
70
71     return 0;
72 }

```

## 5.2 Pure virtual function

```
1  #include <iostream>
2  using namespace std;
3
4  class Player {
5  public:
6      virtual void
7      say_hello(void) = 0;
8  };
9
10 class Ram: public Player {
11 public:
12     void
13     say_hello(void)
14     {
15         cout << "Hello from Ram" << endl;
16     }
17 };
18
19 class Shyam: public Player {
20 public:
21     void
22     say_hello(void)
23     {
24         cout << "Hello from Shyam" << endl;
25     }
26 };
27
28 class Hari: public Player {
29 public:
30     void
31     say_hello(void)
32     {
33         cout << "Hello from Hari" << endl;
34     }
35 };
36
37 class Sita: public Player {
38 public:
39     void
40     say_hello(void)
41     {
42         cout << "Hello from Sita" << endl;
43     }
44 };
45
```

```

46  int
47  main(void)
48  {
49      Player *p;
50
51      Ram r;
52      Shyam s;
53      Hari h;
54      Sita si;
55
56      p = &r;
57      p->say_hello();
58
59      p = &s;
60      p->say_hello();
61
62      p = &h;
63      p->say_hello();
64
65      p = &si;
66      p->say_hello();
67
68      return 0;
69  }

```

## 5.3 Virtual destructor

```

1  #include <iostream>
2  using namespace std;
3
4  class foo {
5  public:
6      virtual ~foo(void)
7      {
8          cout << "Foo is destroyed" << endl;
9      }
10 };
11
12 class bar: public foo {
13 public:
14     ~bar(void)
15     {
16         cout << "Bar is destroyed" << endl;
17     }
18 };
19

```



```

20  int
21  main(void)
22  {
23      foo *f = new bar;
24
25      delete f;
26
27      return 0;
28  }

```

## 5.4 Friend class

```

1  #include <iostream>
2  using namespace std;
3
4  class foo {
5  private:
6      int data;
7  public:
8      void
9      set_data(int d)
10     {
11         data = d;
12     }
13
14     friend class bar;
15 };
16
17 class bar {
18 private:
19     int data;
20 public:
21     void
22     print_foo_data(foo f)
23     {
24         cout << f.data << endl;
25     }
26 };
27
28 int
29 main(void)
30 {
31     foo f;
32     bar b;
33
34     f.set_data(5);

```

```

35         b.print_foo_data(f);
36
37         return 0;
38     }

```

## 5.5 Friend Function

```

1  #include <iostream>
2  using namespace std;
3
4  class foo {
5  private:
6      int bar;
7  public:
8      friend void
9      set_bar(foo &f, int b);
10
11     friend void
12     print_bar(foo f);
13 };
14
15 void
16 set_bar(foo &f, int b)
17 {
18     f.bar = b;
19 }
20
21 void
22 print_bar(foo f)
23 {
24     cout << f.bar << endl;
25 }
26
27 int
28 main(void)
29 {
30     foo f;
31
32     set_bar(f, 14);
33     print_bar(f);
34 }

```

## 5.6 Static function

```
1  #include <iostream>
2  using namespace std;
3
4  class foo {
5  private:
6      int data;
7      static int foo_count;
8  public:
9      foo(void)
10     {
11         foo_count++;
12     }
13
14     static void
15     print_foo_count(void)
16     {
17         cout << foo_count << endl;
18     }
19 };
20
21 int foo::foo_count = 0;
22
23 int
24 main(void)
25 {
26     foo f1, f2, f3;
27
28     foo::print_foo_count();
29
30     return 0;
31 }
```

# Chapter 6

## Function Templates and Exception Handling

### 6.1 Function templates

```
1  #include <iostream>
2  using namespace std;
3
4  template <class T>
5  T my_max(T x, T y)
6  {
7      return (x > y) ? x : y;
8  }
9
10 int
11 main(void)
12 {
13     cout << my_max(30, 20) << endl;
14     cout << my_max(2.5, 10.3) << endl;
15     cout << my_max('c', 'x') << endl;
16
17     return 0;
18 }
```

### 6.2 Function templates with multiple arguments

```
1  #include <iostream>
2  using namespace std;
3
4  template <class T, class U>
5  T my_max(T x, U y)
6  {
```

```

7         return (x > y) ? x : y;
8     }
9
10    int
11    main(void)
12    {
13        cout << my_max(5, 3.4) << endl;
14        cout << my_max(10.3, 4) << endl;
15        cout << my_max(100, 'y') << endl;
16    }

```

## 6.3 Class template

```

1  #include <iostream>
2  using namespace std;
3
4  template <class T>
5  class foo {
6  private:
7      T data;
8  public:
9      void
10     set_data(T d)
11     {
12         data = d;
13     }
14
15     void
16     print_data(void)
17     {
18         cout << data << endl;
19     }
20 };
21
22 int
23 main(void)
24 {
25     foo<int> f;
26
27     f.set_data(4.2);
28     f.print_data();
29 }

```

## 6.4 Inheritance in template class

```
1  #include <iostream>
2  using namespace std;
3
4  template <class T>
5  class foo {
6  private:
7      T data;
8  public:
9      void
10     set_data(T d)
11     {
12         data = d;
13     }
14
15     void
16     print_data(void)
17     {
18         cout << data << endl;
19     }
20 };
21
22 template <class T>
23 class bar: public foo<T> {
24 private:
25     T data;
26 public:
27     void
28     set_data(T d1, T d2)
29     {
30         data = d1;
31         foo<T>::set_data(d2);
32     }
33
34     void
35     print_data(void)
36     {
37         cout << data << endl;
38         foo<T>::print_data();
39     }
40 };
41
42 int
43 main(void)
44 {
45     bar<int> b1;
```

```

46         bar<double> b2;
47
48         b1.set_data(3, 5);
49         b2.set_data(1.3, 6.2);
50
51         b1.print_data();
52         b2.print_data();
53     }

```

## 6.5 Exception Handling

```

1  #include <iostream>
2  using namespace std;
3
4  int
5  main(void)
6  {
7      int a, b;
8
9      a = 4;
10     b = 0;
11
12     try {
13         if (b == 0)
14             throw b;
15         else
16             cout << "a / b = " << a/b << endl;
17     }
18     catch(int x)
19     {
20         cout << "Divided by zero" << endl;
21     }
22 }

```

## 6.6 Multiple catch blocks for a single try block

```

1  #include <iostream>
2  using namespace std;
3
4  void
5  test(int x)
6  {
7      try {
8          if (x > 0) {
9              throw x;

```

```

10         } else {
11             throw 'x';
12         }
13     } catch(int a) {
14         cout << "Its a digit: " << a << endl;
15     } catch(char a) {
16         cout << "Its a character: " << a << endl;
17     }
18 }
19
20 int
21 main(void)
22 {
23     test(4);
24     test(0);
25 }

```

## 6.7 Catch all exceptions

```

1  #include <iostream>
2  using namespace std;
3
4  int
5  main(void)
6  {
7      int num1, num2, result;
8      char op;
9
10     try {
11         cout << "Enter the first number: ";
12         cin >> num1;
13
14         cout << "Enter the operator: ";
15         cin >> op;
16
17         cout << "Enter the second number: ";
18         cin >> num2;
19
20         switch (op) {
21             case '+':
22                 result = num1 + num2;
23                 break;
24             case '-':
25                 result = num1 - num2;
26                 break;
27             case '*':

```



```

28         result = num1 * num2;
29         break;
30     case '/':
31         if (num2 == 0) {
32             throw num2;
33         }
34         result = num1 / num2;
35         break;
36     default:
37         throw op;
38     }
39
40     cout << result << endl;
41 } catch(...) {
42     cout << "Oh no! Exception occurred!" << endl;
43 }
44 }

```

# Chapter 7

## File handling and Streams

### 7.1 Read contents of a file character by character

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4
5  int
6  main(void)
7  {
8      char ch;
9      ifstream fin;
10
11     fin.open("text");
12     while (!fin.eof()) {
13         fin.get(ch);
14         cout << ch;
15     }
16
17     fin.close();
18     return 0;
19 }
```

### 7.2 Writing as well as reading from the file

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4
5  int
6  main(void)
7  {
```

```

8      char name[100];
9      int age;
10
11     /* = PUTTING CONTENTS INTO THE FILE = */
12     ofstream outfile;
13     outfile.open("text");      /* Opening the file we want to write to */
14
15     cout << "Enter your name: ";
16     cin.getline(name, 100);    /* same as 'cin >> name;' */
17     outfile << name << endl;  /* Writing the name with a new line into the
↪ file */
18
19     cout << "Enter your age: ";
20     cin >> age;                /* we cannot use the getline method because
↪ age is an int */
21     outfile << age << endl;    /* Appending the age and a new line into the
↪ file */
22
23     outfile.close();           /* Closing the opened file stream */
24
25     /* = READING CONTENTS FROM THE FILE = */
26     ifstream infile;
27     infile.open("text");      /* Opening the file we want to read from */
28
29     infile.getline(name, 100); /* same as 'infile >> name;' */
30     cout << name << endl;
31
32     infile >> age;             /* read the age */
33     cout << age << endl;
34
35     infile.close();           /* Closing the opened file stream */
36
37     return 0;
38 }

```

## 7.3 Write and Read to/from a single object

```

1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4
5  class foo {
6  private:
7      int data;
8  public:
9      void

```

```

10     set_data(int d)
11     {
12         data = d;
13     }
14
15     void
16     print_data(void)
17     {
18         cout << data << endl;
19     }
20 };
21
22 int
23 main(void)
24 {
25     foo f, f2;
26
27     /* = WRITE OUT THE OBJECT TO THE FILE = */
28     ofstream fout;
29
30     fout.open("text");
31
32     f.set_data(4);
33     fout.write((char *)&f, sizeof(foo));
34
35     fout.close();
36
37     /* = READ ABOUT THE OBJECT FROM THE FILE = */
38     ifstream fin;
39
40     fin.open("text");
41
42     fin.read((char *)&f2, sizeof(foo));
43     f2.print_data();
44
45     fin.close();
46 }

```

## 7.4 Write multiple objects

```

1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4
5  class foo {
6  private:

```

```

7         int data;
8     public:
9         void
10        take_data(void)
11        {
12            cout << "Enter a data: ";
13            cin >> data;
14        }
15
16        void
17        print_data(void)
18        {
19            cout << data << endl;
20        }
21    };
22
23    int
24    main(void)
25    {
26        foo f;
27        ofstream fout;
28
29        fout.open("text");
30
31        for (int i = 0; i < 5; i++) {
32            f.take_data();
33            fout.write((char *)&f, sizeof(foo));
34        }
35
36        fout.close();
37        return 0;
38    }

```

## 7.5 Read multiple objects

```

1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4
5  class foo {
6  private:
7      int data;
8  public:
9      void
10     set_data(void)
11     {

```

```

12         cout << "Enter a data: " << endl;
13         cin >> data;
14     }
15
16     void
17     print_data(void)
18     {
19         cout << data << endl;
20     }
21 };
22
23 int
24 main(void)
25 {
26     foo f;
27     ifstream fin;
28
29     fin.open("writtentext");
30
31     for (int i = 0; i < 5; i++) {
32         fin.read((char *)&f, sizeof(foo));
33
34         f.print_data();
35     }
36
37     fin.close();
38 }

```

## 7.6 Overloading insertion operator

```

1  #include <iostream>
2  using namespace std;
3
4  class foo {
5  private:
6      int data;
7  public:
8      void
9      set_data(int d)
10     {
11         data = d;
12     }
13
14     friend ostream &
15     operator <<(ostream &o, foo f);
16 };

```

```

17
18 ostream &
19 operator <<(ostream &os, foo f)
20 {
21     os << f.data;
22
23     return os;
24 }
25
26 int
27 main(void)
28 {
29     foo f;
30
31     f.set_data(4);
32
33     cout << f << endl;
34 }

```

## 7.7 Overloading extraction operator

```

1  #include <iostream>
2  using namespace std;
3
4  class foo {
5  private:
6      int data;
7  public:
8      void
9      print_data(void)
10     {
11         cout << data << endl;
12     }
13
14     friend istream &
15     operator >>(istream &is, foo &f);
16 };
17
18 istream &
19 operator >>(istream &is, foo &f)
20 {
21     is >> f.data;
22
23     return is;
24 }
25

```

```

26  int
27  main(void)
28  {
29      foo f;
30
31      cin >> f;
32      f.print_data();
33  }

```

## 7.8 Overloading both insertion and extraction operator

```

1  #include <iostream>
2  using namespace std;
3
4  class foo {
5  private:
6      int data;
7  public:
8      friend ostream &
9      operator <<(ostream &os, foo f);
10
11     friend istream &
12     operator >>(istream &is, foo &f);
13 };
14
15 ostream &
16 operator <<(ostream &os, foo f)
17 {
18     os << f.data;
19
20     return os;
21 }
22
23 istream &
24 operator >>(istream &is, foo &f)
25 {
26     is >> f.data;
27
28     return is;
29 }
30
31 int
32 main(void)
33 {
34     foo f;
35

```



```
36     cin >> f;  
37     cout << f << endl;  
38 }
```

# Chapter 8

## Extra

### 8.1 Reverse string

```
1  #include <iostream>
2  using namespace std;
3
4  string
5  reverse(string msg)
6  {
7      string rev;
8
9      for (int i = 0; i < msg.length(); i++) {
10         rev += msg[msg.length() - 1 - i];
11     }
12
13     return rev;
14 }
15
16 int
17 main(void)
18 {
19     string msg, rev_msg;
20
21     msg = "foobar";
22     rev_msg = reverse(msg);
23
24     cout << rev_msg << endl;
25 }
```

## 8.2 Stack implementation using templates

```
1  #include <iostream>
2  using namespace std;
3
4  template <class T>
5  class my_stack {
6  private:
7      T data[999];
8      int cur_index;
9  public:
10     my_stack(void)
11     {
12         cur_index = 0;
13     }
14
15     void
16     push(T d)
17     {
18         data[cur_index++] = d;
19     }
20
21     T
22     pop(void)
23     {
24         return data[--cur_index];
25     }
26 };
27
28 int
29 main(void)
30 {
31     my_stack<int> numbers;
32     my_stack<string> strings;
33
34     numbers.push(5);
35     numbers.push(6);
36     numbers.push(7);
37
38     cout << numbers.pop() << endl;
39     cout << numbers.pop() << endl;
40     cout << numbers.pop() << endl;
41
42     strings.push("foo");
43     strings.push("bar");
44     strings.push("buzz");
45 }
```

```

46         cout << strings.pop() << endl;
47         cout << strings.pop() << endl;
48         cout << strings.pop() << endl;
49     }

```

## 8.3 2078 - 1

```

1  #include<iostream>
2  using namespace std;
3
4  class Account {
5  private:
6      string acc_no;
7      int balance;
8      static int min_balance;
9
10 public:
11     void
12     set_values(void)
13     {
14         string acc_no;
15         int balance;
16
17         cout << "Enter the account number: ";
18         cin >> acc_no;
19
20         cout << "Enter the balance: ";
21         cin >> balance;
22
23         this->acc_no = acc_no;
24         this->balance = balance;
25
26         if (min_balance == 0 || min_balance > balance) {
27             min_balance = balance;
28         }
29     }
30
31     void
32     print_values(void)
33     {
34         cout << "Account no = " << acc_no << ", Balance = " << balance <<
↵ endl;
35     }
36
37     static void
38     print_min_balance(void)

```

```

39         {
40             cout << min_balance << endl;
41         }
42     };
43
44     int Account::min_balance = 0;
45
46     int
47     main(void)
48     {
49         Account a[5];
50
51         for (int i = 0; i < 5; i++) {
52             a[i].set_values();
53         }
54
55         for (int i = 0; i < 5; i++) {
56             a[i].print_values();
57         }
58
59         Account::print_min_balance();
60     }

```

## 8.4 2076 - 1

```

1  #include <iostream>
2  using namespace std;
3
4  class Teacher {
5  private:
6      string tid;
7      string subject;
8  public:
9      void
10     set_values(void)
11     {
12         cout << "Enter the teacher id: ";
13         cin >> tid;
14
15         cout << "Enter the teacher subject: ";
16         cin >> subject;
17     }
18
19     void
20     print_values(void)
21     {

```

```

22         cout << "TID = " << tid << ", Subject = " << subject << endl;
23     }
24 };
25
26 class Staff {
27 private:
28     string sid;
29     string position;
30 public:
31     void
32     set_values(void)
33     {
34         cout << "Enter the staff id: ";
35         cin >> sid;
36
37         cout << "Enter the staff position: ";
38         cin >> position;
39     }
40
41     void
42     print_values(void)
43     {
44         cout << "SID = " << sid << ", Position = " << position << endl;
45     }
46 };
47
48 class Coordinator: public Teacher, public Staff {
49 private:
50     string department;
51 public:
52     void
53     set_values(void)
54     {
55         Staff::set_values();
56         Teacher::set_values();
57
58         cout << "Enter the department: ";
59         cin >> department;
60     }
61
62     void
63     print_values(void)
64     {
65         Staff::print_values();
66         Teacher::print_values();
67
68         cout << "Department = " << department << endl;
69     }

```

```
70 };
71
72 int
73 main(void)
74 {
75     Coordinator c1, c2;
76
77     c1.set_values();
78     c2.set_values();
79
80     c1.print_values();
81     c2.print_values();
82
83     return 0;
84 }
```