Contents

Chapter 2

Basics of C++ Programming

2.1 Namespace

```
#include <iostream>

using namespace std;

namespace foo {
int bar = 10;
}

int
main(void)
{
        cout << foo::bar << endl;

        return 0;
}
```

2.2 Manipulators

```
#include <iostream>
#include <iomanip>

using namespace std;

int
main(void)
{
        float foo;
        string bar;

```

```
        foo = 123.456;
        bar = "buzz";

        cout << setprecision(4) << foo << endl;
        cout << setw(10) << bar << endl;

        return 0;
}

/*
Output:

123.5
      buzz
*/
```

2.3 Dynamic memory allocation

```
#include <iostream>

using namespace std;

int
main(void)
{
        int *foo;

        foo = new int;
        *foo = 5;

        cout << *foo << endl;

        delete foo;

        return 0;
}
```

2.4 Functions

2.4.1 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.

```
#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 }
```

2.4.2 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 }
```

2.4.3 Function overloading – Different types of arguments

```cpp
#include <iostream>

using namespace std;

int
add(int num1, int num2)
{
        return num1 + num2;
}

float
add(float num1, float num2)
{
        return num1 + num2;
}

int
main(void)
{
        cout << add(5, 2) << endl;
        cout << add((float)5.5, (float)3.2) << endl;

        return 0;
}
```

2.4.4 Function overloading – Different number of arguments

```cpp
#include <iostream>

using namespace std;

int
add(int num1, int num2)
{
        return num1 + num2;
}

int
add(int num1, int num2, int num3)
{
        return num1 + num2 + num3;
}

int
main(void)
```

```
{
        cout << add(4, 2) << endl;
        cout << add(6, 4, 2) << endl;

        return 0;
}
```

2.4.5 Pass by value

```
#include <iostream>

using namespace std;

void
exchange(int num1, int num2)
{
        int temp_num;

        temp_num = num1;
        num1 = num2;
        num2 = temp_num;
}

int
main(void)
{
        int num1, num2;

        num1 = 5;
        num2 = 7;

        cout << "Before exchange: " << num1 << "-" << num2 << endl;

        exchange(num1, num2);

        cout << "After exchange: " << num1 << "-" << num2 << endl;

        return 0;
}
```

2.4.6 Pass by reference

```
#include <iostream>

using namespace std;

```

```
void
exchange(int &num1, int &num2)
{
        int temp_num;

        temp_num = num1;
        num1 = num2;
        num2 = temp_num;
}

int
main(void)
{
        int num1, num2;

        num1 = 5;
        num2 = 7;

        cout << "Before exchange: " << num1 << "-" << num2 << endl;

        exchange(num1, num2);

        cout << "After exchange: " << num1 << "-" << num2 << endl;

        return 0;
}
```

2.4.7 Pass by pointer

```
#include <iostream>

using namespace std;

void
exchange(int *num1, int *num2)
{
        int temp_num;

        temp_num = *num1;
        *num1 = *num2;
        *num2 = temp_num;
}

int
main(void)
{
        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  }
```

2.4.8 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  }
```

2.4.9 Return by reference

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

```
largest(int &num1, int &num2)
{
        return num1 > num2 ? num1 : num2;
}

int
main(void)
{
        int num1, num2;

        num1 = 5;
        num2 = 7;

        cout << num1 << "-" << num2 << endl;

        largest(num1, num2) = 0;

        cout << num1 << "-" << num2 << endl;

        return 0;
}
```

2.4.10 Return by pointer

```
#include <iostream>

using namespace std;

int *
largest(int *num1, int *num2)
{
        return *num1 > *num2 ? num1 : num2;
}

int
main(void)
{
        int num1, num2;

        num1 = 5;
        num2 = 7;

        cout << num1 << "-" << num2 << endl;

        *(largest(&num1, &num2)) = 0;

        cout << num1 << "-" << num2 << endl;
```

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

Chapter 3

Classes and Objects

3.1 Constructors

3.1.1 Default Constructor

```
#include <iostream>

using namespace std;

class foo {
public:
        foo()
        {
                cout << "Foo object created" << endl;
        }
};

int
main(void)
{
        foo f;

        return 0;
}
```

3.1.2 Parameterized Constructor

```
#include <iostream>

using namespace std;

class foo {
private:
```

```
        int bar;
public:
        foo()
        {
                cout << "No parameter was given" << endl;
        }

        foo(int b)
        {
                cout << "Bar was given" << endl;
                bar = b;
        }
};

int
main(void)
{
        foo f1, f2(5);

        return 0;
}
```

3.1.3 Copy constructor

```
#include <iostream>

using namespace std;

class foo {
private:
        int bar;
public:
        foo(int b)
        {
                cout << "A value to bar is given" << endl;
                bar = b;
        }

        foo(foo &f)
        {
                cout << "A foo object instance is given" << endl;
                bar = f.bar;
        }
};

int
main(void)
```

```
{
        foo f1(5), f2(f1);

        return 0;
}
```

3.1.4 Default copy constructor

```
#include <iostream>

using namespace std;

class foo {
private:
        int bar;
public:
        foo(int b)
        {
                cout << "A value to bar is given" << endl;
                bar = b;
        }

        void
        print_bar(void)
        {
                cout << bar << endl;
        }
};

int
main(void)
{
        foo f1(5), f2(f1);

        f1.print_bar();
        f2.print_bar();

        return 0;
}

/*
Output:

A value to bar is given
5
5
*/
```

3.2 Destructor

```
#include <iostream>

using namespace std;

class foo {
public:
        foo()
        {
                cout << "Foo object is created" << endl;
        }

        ~foo()
        {
                cout << "Foo object is destroyed" << endl;
        }
};

int
main(void)
{
        foo f;

        return 0;
}
```

3.3 Objects as Function Arguments

3.3.1 Pass by value

```
#include <iostream>

using namespace std;

class distances {
private:
        int feet;
        int inches;

        void
        recalc(void)
        {
                feet = feet + inches/12;
                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
```

```
/*
Output:

Feet: 4, Inches: 10
Feet: 7, Inches: 2
*/
```

3.3.2 Pass by reference

```cpp
#include <iostream>

using namespace std;

class foo {
private:
        int bar;
public:
        void
        set_bar(int b)
        {
                bar = b;
        }

        /* This function changes the value of `bar` in the passed `foo` object
         * with the value of `bar` in the current object. */
        void
        change_bar(foo &f)
        {
                f.bar = bar;
        }

        void
        print_bar(void)
        {
                cout << bar << endl;
        }
};

int
main(void)
{
        foo b1, b2;

        b1.set_bar(5);
        b1.change_bar(b2);

        b1.print_bar();
        b2.print_bar();

        return 0;
}

/*
Output:
5
5
*/
```

3.3.3 Pass by pointer

```
#include <iostream>

using namespace std;

class foo {
private:
        int bar;
public:
        void
        set_bar(int b)
        {
                bar = b;
        }

        /* This function changes the value of `bar` in the passed `foo` object
         * with the value of `bar` in the current object. */
        void
        change_bar(foo *f)
        {
                f->bar = bar;
        }

        void
        print_bar(void)
        {
                cout << bar << endl;
        }
};

int
main(void)
{
        foo b1, b2;

        b1.set_bar(5);
        b1.change_bar(&b2);

        b1.print_bar();
        b2.print_bar();

        return 0;
}

/*
Output:
5
```

```
5
*/
```

3.4 Structure

3.4.1 Showing all the extensions to structure

```
#include <iostream>

using namespace std;

struct foo {
private:
        int bar;
public:
        void
        set_bar(int b)
        {
                bar = b;
        }

        void
        print_bar(void)
        {
                cout << bar << endl;
        }
};

int
main(void)
{
        foo f;

        f.set_bar(5);
        f.print_bar();

        return 0;
}
```

3.5 Static data members in class

```
#include <iostream>
using namespace std;

```

```
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 }
```

3.6 Member functions defined outside the class

```cpp
#include <iostream>
using namespace std;

class foo {
private:
        int bar;
public:
        void
        set_bar(int f);

        void
        print_bar(void);
};

void
foo::set_bar(int f)
{
        bar = f;
}

void
foo::print_bar(void)
{
        cout << bar << endl;
}

int
main(void)
{
        foo f;

        f.set_bar(5);
        f.print_bar();

        return 0;
}
```

Chapter 4

Operator Overloading

4.1 Overloading unary operator

Prefix and Postfix

```
#include <iostream>
using namespace std;

class rectangle {
private:
        int length;
        int breadth;

public:
        void
        set_data(int l, int b)
        {
                length  = l;
                breadth = b;
        }

        void
        print_data(void)
        {
                cout << "Length: " << length << ", Breadth: " << breadth
                     << endl;
        }

        /* pre-increment */
        void
        operator ++(void)
        {
                ++length;
                ++breadth;
```

```
        }

        /* post-increment */
        void
        operator ++(int)
        {
                length++;
                breadth++;
        }
};

int
main(void)
{
        rectangle r1, r2;

        r1.set_data(5, 6);
        r2.set_data(7, 3);

        r1++;
        ++r2;

        r1.print_data();
        r2.print_data();

        return 0;
}
```

4.2 Overloading binary operator

Plus

```
#include <iostream>
using namespace std;

class rectangle {
private:
        int length;
        int breadth;

public:
        void
        set_data(int l, int b)
        {
                length  = l;
                breadth = b;
```

```
        }

        void
        print_data(void)
        {
                cout << "Length: " << length << ", Breadth: " << breadth
                     << endl;
        }

        /* plus */
        rectangle
        operator +(rectangle r)
        {
                rectangle temp_r;

                temp_r.length  = length + r.length;
                temp_r.breadth = breadth + r.breadth;

                return temp_r;
        }
};

int
main(void)
{
        rectangle r1, r2, r3;

        r1.set_data(5, 6);
        r2.set_data(7, 3);

        r3 = r1 + r2;
        r3.print_data();

        return 0;
}
```

Comparison

```
#include <iostream>
using namespace std;

class length {
private:
        int data;
public:
        void
        set_data(int d)
```

```
        {
                data = d;
        }

        void
        print_data(void)
        {
                cout << "Length = " << data << endl;
        }

        int
        operator >(length l)
        {
                return l.data > data;
        }
};

int
main(void)
{
        length r1, r2;

        cout << (r1 > r2 ? "r1 is greater" : "r2 is greater") << endl;

        return 0;
}
```

Assignment

```
#include <iostream>
using namespace std;

class length {
private:
        int data;
public:
        void
        set_data(int d)
        {
                data = d;
        }

        void
        print_data(void)
        {
                cout << data << endl;
        }
```

```
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  }
```

4.3 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  }
```

4.4 Data Conversion

4.4.1 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 }
```

4.4.2 From basic to user defined

```
#include <iostream>
using namespace std;

class distances {
private:
        int feet;
        int inches;
public:
        distances(int i)
        {
                feet = i / 12;
                inches = i % 12;
        }

        void
        print_data(void)
        {
                cout << "Feet = " << feet << ", Inches = " << inches << endl;
        }
};

int
main(void)
{
        distances d(14);

        d.print_data();

        return 0;
}
```

4.4.3 From user defined to basic

```
#include <iostream>
using namespace std;

class height {
private:
        int feet;
        int inches;
public:
        height(int f, int i)
        {
                feet = f;
                inches = i;
```

```
        }

        operator int(void)
        {
                int temp_i;

                temp_i = (feet * 12) + inches;

                return temp_i;
        }
};

int
main(void)
{
        height h(3, 7);
        int inches;

        inches = (int)h;

        cout << inches << endl;

        return 0;
}
```

4.4.4 From user defined to user defined

Routine in the source class

```
#include <iostream>
using namespace std;

class length_centimeter {
private:
        int centimeter;
public:
        void
        set_centimeter(int c)
        {
                centimeter = c;
        }

        void
        print_centimeter(void)
        {
                cout << centimeter << endl;
        }
```

```
};

class length_meter {
private:
        int meter;
public:
        void
        set_meter(int m)
        {
                meter = m;
        }

        operator length_centimeter(void)
        {
                length_centimeter temp_c;

                temp_c.set_centimeter(meter * 100);

                return temp_c;
        }
};

int
main(void)
{
        length_centimeter cm;
        length_meter m;

        m.set_meter(3);
        cm = (length_centimeter)m;

        cm.print_centimeter();

        return 0;
}
```

Routine in destination class

```
#include <iostream>
using namespace std;

class length_meter {
private:
        int meter;
public:
        void
        set_meter(int m)
```

```
        {
                meter = m;
        }

        int
        get_meter(void)
        {
                return meter;
        }
};

class length_centimeter {
private:
        int centimeter;
public:
        length_centimeter(length_meter m)
        {
                centimeter = m.get_meter() * 100;
        }

        void
        print_centimeter(void)
        {
                cout << centimeter << endl;
        }
};

int
main(void)
{
        length_meter m;

        m.set_meter(3);

        length_centimeter cm(m);
        cm.print_centimeter();

        return 0;
}
```

ALTERNATIVE CODE

```
#include <iostream>
using namespace std;

class length_meter {
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

```
#include <iostream>
using namespace std;

class foo {
public:
        int data;
};

class bar {
public:
        int data;

        bar(void)
        {}

        /* routine in source */
        operator
        foo(void)
        {
                foo temp_f;

                temp_f.data = data;

                return temp_f;
        }

        /* routine in destination */
        bar(foo f)
        {
                data = f.data;
        }
};

int
main(void)
{
        foo f;
        bar b;

        /* routine in source */
        b.data = 5;
        f = (foo)b;
        cout << f.data << endl;

        /* routine in destination */
        f.data = 6;
```

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

Chapter 5

Inheritance

5.1 Overriding member functions

```
#include <iostream>
using namespace std;

class foo {
public:
        void
        say_hello(void)
        {
                cout << "Hello from foo" << endl;
        }
};

class bar: public foo {
public:
        void
        say_hello(void)
        {
                cout << "Hello from bar" << endl;
        }
};

int
main(void)
{
        bar b;

        b.foo::say_hello();

        return 0;
}
```

5.2 Types of inheritance

5.2.1 Single

```
#include <iostream>
using namespace std;

class foo {
public:
        foo(void)
        {
                cout << "Class foo initialized" << endl;
        }
};

class bar: public foo {
public:
        bar(void)
        {
                cout << "Class bar initialized" << endl;
        }
};

int
main(void)
{
        bar b;

        return 0;
}
```

5.2.2 Multiple

```
#include <iostream>
using namespace std;

class foo {
public:
        foo(void)
        {
                cout << "Class foo initialized" << endl;
        }
};

class bar {
public:
```

```
        bar(void)
        {
                cout << "Class bar initialized" << endl;
        }
};

class buzz: public foo, public bar {
public:
        buzz(void)
        {
                cout << "Class buzz initialized" << endl;
        }
};

int
main(void)
{
        buzz b;

        return 0;
}
```

5.2.3 Hierarchical

```
#include <iostream>
using namespace std;

class vehicle {
public:
        vehicle(void)
        {
                cout << "Vehicle initialized" << endl;
        }
};

class bike: public vehicle {
public:
        bike(void)
        {
                cout << "Bike initialized" << endl;
        }
};

class car: public vehicle {
public:
        car(void)
        {
```

```
                cout << "Car initialized" << endl;
        }
};

int
main(void)
{
        bike b;
        car c;

        return 0;
}
```

5.2.4 Multilevel

```
#include <iostream>
using namespace std;

class vehicle {
public:
        vehicle(void)
        {
                cout << "Vehicle initialized" << endl;
        }
};

class bike: public vehicle {
public:
        bike(void)
        {
                cout << "Bike initialized" << endl;
        }
};

class xpulse: public bike {
public:
        xpulse(void)
        {
                cout << "Xpulse initialized" << endl;
        }
};

int
main(void)
{
        xpulse x;

```

```
33          return 0;
34  }
```

5.2.5 Hybrid

Jhew lagyo

5.3 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  }
```

5.4 Ambiguity in multipath inheritance

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

```
        int a;
};

class B: virtual public A {
public:
        int b;
};

class C: virtual public A {
public:
        int c;
};

class D: public B, public C {
public:
        int d;
};

int
main(void)
{
        D o;

        o.a = 1;
        o.b = 2;
        o.c = 3;
        o.d = 4;

        cout << o.a << endl;
        cout << o.b << endl;
        cout << o.c << endl;
        cout << o.d << endl;
}
```

5.5 Aggregation

```
#include <iostream>
using namespace std;

class employee {
private:
        string e_id;
        string e_name;
public:
        void
        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 6

Virtual function, Polymorphism and Miscellaneous C++ Features

6.1 Virtual function

```
#include <iostream>
using namespace std;

class Player {
public:
        virtual void
        say_hello(void)
        {
                cout << "Hello from Player" << endl;
        }
};

class Ram: public Player {
public:
        void
        say_hello(void)
        {
                cout << "Hello from Ram" << endl;
        }
};

class Shyam: public Player {
public:
        void
        say_hello(void)
        {
                cout << "Hello from Shyam" << endl;
        }
};
```

```
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  }
```

6.2 Pure virtual function

```
#include <iostream>
using namespace std;

class Player {
public:
        virtual void
        say_hello(void) = 0;
};

class Ram: public Player {
public:
        void
        say_hello(void)
        {
                cout << "Hello from Ram" << endl;
        }
};

class Shyam: public Player {
public:
        void
        say_hello(void)
        {
                cout << "Hello from Shyam" << endl;
        }
};

class Hari: public Player {
public:
        void
        say_hello(void)
        {
                cout << "Hello from Hari" << endl;
        }
};

class Sita: public Player {
public:
        void
        say_hello(void)
        {
                cout << "Hello from Sita" << endl;
        }
};

```

```
int
main(void)
{
        Player *p;

        Ram r;
        Shyam s;
        Hari h;
        Sita si;

        p = &r;
        p->say_hello();

        p = &s;
        p->say_hello();

        p = &h;
        p->say_hello();

        p = &si;
        p->say_hello();

        return 0;
}
```

6.3 Virtual destructor

```
#include <iostream>
using namespace std;

class foo {
public:
        virtual ~foo(void)
        {
                cout << "Foo is destroyed" << endl;
        }
};

class bar: public foo {
public:
        ~bar(void)
        {
                cout << "Bar is destroyed" << endl;
        }
};

```

```
int
main(void)
{
        foo *f = new bar;

        delete f;

        return 0;
}
```

6.4 Friend class

```
#include <iostream>
using namespace std;

class foo {
private:
        int data;
public:
        void
        set_data(int d)
        {
                data = d;
        }

        friend class bar;
};

class bar {
private:
        int data;
public:
        void
        print_foo_data(foo f)
        {
                cout << f.data << endl;
        }
};

int
main(void)
{
        foo f;
        bar b;

        f.set_data(5);
```

```
        b.print_foo_data(f);

        return 0;
}
```

6.5 Friend Function

```
#include <iostream>
using namespace std;

class foo {
private:
        int bar;
public:
        friend void
        set_bar(foo &f, int b);

        friend void
        print_bar(foo f);
};

void
set_bar(foo &f, int b)
{
        f.bar = b;
}

void
print_bar(foo f)
{
        cout << f.bar << endl;
}

int
main(void)
{
        foo f;

        set_bar(f, 14);
        print_bar(f);
}
```

6.6 Static function

```
#include <iostream>
using namespace std;

class foo {
private:
        int data;
        static int foo_count;
public:
        foo(void)
        {
                foo_count++;
        }

        static void
        print_foo_count(void)
        {
                cout << foo_count << endl;
        }
};

int foo::foo_count = 0;

int
main(void)
{
        foo f1, f2, f3;

        foo::print_foo_count();

        return 0;
}
```

Chapter 7

Function Templates and Exception Handling

7.1 Function templates

```
#include <iostream>
using namespace std;

template <class T>
T my_max(T x, T y)
{
        return (x > y) ? x : y;
}

int
main(void)
{
        cout << my_max(30, 20) << endl;
        cout << my_max(2.5, 10.3) << endl;
        cout << my_max('c', 'x') << endl;

        return 0;
}
```

7.2 Function templates with multiple arguments

```
#include <iostream>
using namespace std;

template <class T, class U>
T my_max(T x, U y)
{
```

```
        return (x > y) ? x : y;
}

int
main(void)
{
        cout << my_max(5, 3.4) << endl;
        cout << my_max(10.3, 4) << endl;
        cout << my_max(100, 'y') << endl;
}
```

7.3 Class template

```
#include <iostream>
using namespace std;

template <class T>
class foo {
private:
        T data;
public:
        void
        set_data(T d)
        {
                data = d;
        }

        void
        print_data(void)
        {
                cout << data << endl;
        }
};

int
main(void)
{
        foo<int> f;

        f.set_data(4.2);
        f.print_data();
}
```

7.4 Inheritance in template class

```
#include <iostream>
using namespace std;

template <class T>
class foo {
private:
        T data;
public:
        void
        set_data(T d)
        {
                data = d;
        }

        void
        print_data(void)
        {
                cout << data << endl;
        }
};

template <class T>
class bar: public foo<T> {
private:
        T data;
public:
        void
        set_data(T d1, T d2)
        {
                data = d1;
                foo<T>::set_data(d2);
        }

        void
        print_data(void)
        {
                cout << data << endl;
                foo<T>::print_data();
        }
};

int
main(void)
{
        bar<int> b1;
```

```
        bar<double> b2;

        b1.set_data(3, 5);
        b2.set_data(1.3, 6.2);

        b1.print_data();
        b2.print_data();
}
```

7.5 Exception Handling

```
#include <iostream>
using namespace std;

int
main(void)
{
        int a, b;

        a = 4;
        b = 0;

        try {
                if (b == 0)
                        throw b;
                else
                        cout << "a / b = " << a/b << endl;
        }
        catch(int x)
        {
                cout << "Divided by zero" << endl;
        }
}
```

7.6 Multiple catch blocks for a single try block

```
#include <iostream>
using namespace std;

void
test(int x)
{
        try {
                if (x > 0) {
                        throw x;
```

```
                } else {
                        throw 'x';
                }
        } catch(int a) {
                cout << "Its a digit: " << a << endl;
        } catch(char a) {
                cout << "Its a character: " << a << endl;
        }
}

int
main(void)
{
        test(4);
        test(0);
}
```

7.7 Catch all exceptions

```
#include <iostream>
using namespace std;

int
main(void)
{
        int num1, num2, result;
        char op;

        try {
                cout << "Enter the first number: ";
                cin >> num1;

                cout << "Enter the operator: ";
                cin >> op;

                cout << "Enter the second number: ";
                cin >> num2;

                switch (op) {
                case '+':
                        result = num1 + num2;
                        break;
                case '-':
                        result = num1 - num2;
                        break;
                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 8

File handling and Streams

8.1 Read contents of a file character by character

```
#include <iostream>
#include <fstream>
using namespace std;

int
main(void)
{
        char ch;
        ifstream fin;

        fin.open("text");
        while (!fin.eof()) {
                fin.get(ch);
                cout << ch;
        }

        fin.close();
        return 0;
}
```

8.2 Writing as well as reading from the file

```
#include <iostream>
#include <fstream>
using namespace std;

int
main(void)
{
```

```
        char name[100];
        int age;

        /* = PUTTING CONTENTS INTO THE FILE = */
        ofstream outfile;
        outfile.open("text");       /* Opening the file we want to write to */

        cout << "Enter your name: ";
        cin.getline(name, 100);     /* same as 'cin >> name;' */
        outfile << name << endl;    /* Writing the name with a new line into the
    ↪  file */

        cout << "Enter your age: ";
        cin >> age;                 /* we cannot use the getline method because
    ↪  age is an int */
        outfile << age << endl;     /* Appending the age and a new line into the
    ↪  file */

        outfile.close();            /* Closing the opened file stream */

        /* = READING CONTENTS FROM THE FILE = */
        ifstream infile;
        infile.open("text");        /* Opening the file we want to read from */

        infile.getline(name, 100);  /* same as 'infile >> name;' */
        cout << name << endl;

        infile >> age;              /* read the age */
        cout << age << endl;

        infile.close();             /* Closing the opened file stream */

        return 0;
}
```

8.3 Write and Read to/from a single object

```
#include <iostream>
#include <fstream>
using namespace std;

class foo {
private:
        int data;
public:
        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  }
```

8.4 Write multiple objects

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

```
        int data;
public:
        void
        take_data(void)
        {
                cout << "Enter a data: ";
                cin >> data;
        }

        void
        print_data(void)
        {
                cout << data << endl;
        }
};

int
main(void)
{
        foo f;
        ofstream fout;

        fout.open("text");

        for (int i = 0; i < 5; i++) {
                f.take_data();
                fout.write((char *)&f, sizeof(foo));
        }

        fout.close();
        return 0;
}
```

8.5 Read multiple objects

```
#include <iostream>
#include <fstream>
using namespace std;

class foo {
private:
        int data;
public:
        void
        set_data(void)
        {
```

```
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  }
```

8.6 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  };
```

```

ostream &
operator <<(ostream &os, foo f)
{
        os << f.data;

        return os;
}

int
main(void)
{
        foo f;

        f.set_data(4);

        cout << f << endl;
}
```

8.7 Overloading extraction operator

```
#include <iostream>
using namespace std;

class foo {
private:
        int data;
public:
        void
        print_data(void)
        {
                cout << data << endl;
        }

        friend istream &
        operator >>(istream &is, foo &f);
};

istream &
operator >>(istream &is, foo &f)
{
        is >> f.data;

        return is;
}

```

```
int
main(void)
{
        foo f;

        cin >> f;
        f.print_data();
}
```

8.8 Overloading both insertion and extraction operator

```
#include <iostream>
using namespace std;

class foo {
private:
        int data;
public:
        friend ostream &
        operator <<(ostream &os, foo f);

        friend istream &
        operator >>(istream &is, foo &f);
};

ostream &
operator <<(ostream &os, foo f)
{
        os << f.data;

        return os;
}

istream &
operator >>(istream &is, foo &f)
{
        is >> f.data;

        return is;
}

int
main(void)
{
        foo f;

```

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

Chapter 9

Extra

9.1 Reverse string

```
#include <iostream>
using namespace std;

string
reverse(string msg)
{
        string rev;

        for (int i = 0; i < msg.length(); i++) {
                rev += msg[msg.length() - 1 - i];
        }

        return rev;
}

int
main(void)
{
        string msg, rev_msg;

        msg = "foobar";
        rev_msg = reverse(msg);

        cout << rev_msg << endl;
}
```

9.2 Stack implementation using templates

```
#include <iostream>
using namespace std;

template <class T>
class my_stack {
private:
        T data[999];
        int cur_index;
public:
        my_stack(void)
        {
                cur_index = 0;
        }

        void
        push(T d)
        {
                data[cur_index++] = d;
        }

        T
        pop(void)
        {
                return data[--cur_index];
        }
};

int
main(void)
{
        my_stack<int> numbers;
        my_stack<string> strings;

        numbers.push(5);
        numbers.push(6);
        numbers.push(7);

        cout << numbers.pop() << endl;
        cout << numbers.pop() << endl;
        cout << numbers.pop() << endl;

        strings.push("foo");
        strings.push("bar");
        strings.push("buzz");

```

```
        cout << strings.pop() << endl;
        cout << strings.pop() << endl;
        cout << strings.pop() << endl;
}
```

9.3 2078 - 1

```
#include<iostream>
using namespace std;

class Account {
private:
        string acc_no;
        int balance;
        static int min_balance;

public:
        void
        set_values(void)
        {
                string acc_no;
                int balance;

                cout << "Enter the account number: ";
                cin >> acc_no;

                cout << "Enter the balance: ";
                cin >> balance;

                this->acc_no = acc_no;
                this->balance = balance;

                if (min_balance == 0 || min_balance > balance) {
                        min_balance = balance;
                }
        }

        void
        print_values(void)
        {
                cout << "Account no = " << acc_no << ", Balance = " << balance <<
 ↪  endl;
        }

        static void
        print_min_balance(void)
```

```
        {
                cout << min_balance << endl;
        }
};

int Account::min_balance = 0;

int
main(void)
{
        Account a[5];

        for (int i = 0; i < 5; i++) {
                a[i].set_values();
        }

        for (int i = 0; i < 5; i++) {
                a[i].print_values();
        }

        Account::print_min_balance();
}
```

9.4 2076 - 1

```
#include <iostream>
using namespace std;

class Teacher {
private:
        string tid;
        string subject;
public:
        void
        set_values(void)
        {
                cout << "Enter the teacher id: ";
                cin >> tid;

                cout << "Enter the teacher subject: ";
                cin >> subject;
        }

        void
        print_values(void)
        {
```

```
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  }
```