Overloading the extraction(>>) and insertion(<<) operator

Overloading the extraction(>>) and insertion(<<) operator

```
friend ostream & operator << ( ostream &out, class_type   obj )
{
        // statements..
        return out;
}
```

- The first parameter to the function is a reference to the output stream. The second parameter is the object being inserted.

```
friend istream & operator >> ( istream &in, class_type   obj )
{
        // statements..
        return in;
}
```

```
3   class Complex
4   {
5   private:
6       int real, imag;
7   public:
8       Complex(int r = 0, int i =0)
9       {  real = r;    imag = i; }
10
11      friend ostream & operator << (ostream &, Complex &);
12      friend istream & operator >> (istream &, Complex &);
13  };
```

```cpp
ostream & operator << (ostream &out,  Complex &C)
{
    out << C.real;
    out << "+i" << C.imag << endl;
    return out;
}

istream & operator >> (istream &in,  Complex &C)
{
    cout << "Enter Real Part ";
    in >> C.real;
    cout << "Enter Imaginary Part ";
    in >> C.imag;
    return in;
}
```

```
 Enter 2 complex numbers:

Enter Real Part 2
Enter Imaginary Part 1

Enter Real Part 3
Enter Imaginary Part 4
The complex numbers are:
2+i1

3+i4
```

```cpp
int main()
{
    Complex c1,c2;
    cout<<"\n Enter 2 complex numbers: \n";
    cin >>c1>>c2;
    cout << "The complex numbers are: \n";
    cout << c1<<"\n"<<c2;
}
```

Question

```
37  int main()
38  {
39      int r , c;
40      cout<<"\nEnter the dimension of the 1st Matrix: ";
41      cin >> r >> c;
42      Matrix M1( r , c );
43      cout<<"\nEnter the elements for the 1st Matrix:\n";
44      cin >> M1;
45      cout<<"\nEnter the dimension of the 2nd Matrix: ";
46      cin >> r >> c;
47      Matrix M2( r , c );
48      cout<<"\nEnter the elements for the 2nd Matrix:";
49      cin >> M2;
50      cout<<"\nThe 1st Matrix:\n";
51      cout << M1;
52      cout << "\nThe 2nd Matrix:\n";
53      cout << M2;
54  }
```

```
3  class Matrix
4  {
5      int row_size, col_size;
6      int **M;
7  public:
8      Matrix( ) { row_size = col_size = 0; }
9      Matrix( int r , int c )
10     {
11         row_size = r;
12         col_size = c;
13         M = new int*[row_size];
14         for( int i = 0 ; i < row_size ; i++ )
15             M[i] = new int[col_size];
16     }
17 friend ostream & operator << ( ostream&, Matrix& );
18 friend istream & operator >> ( istream&, Matrix& );
19 };
```

```
20  istream & operator >> ( istream &in ,  Matrix &Mat )
21  {
22    for(int i = 0 ; i < Mat.row_size; i++ )
23      for(int j = 0; j < Mat.col_size; j++ )
24          in >> Mat.M[i][j];
25      return in;
26  }

27  ostream & operator << ( ostream &out ,  Matrix &MAT )
28  {
29      for(int i = 0 ; i < MAT.row_size; i++ )
30      {
31          for(int j = 0; j < MAT.col_size; j++ )
32              out << MAT.M[i][j]<<"\t";
33          out << "\n";
34      }
35      return out;
36  }
```

```
37  int main()
38  {
39      int r , c;
40      cout<<"\nEnter the dimension of the 1st Matrix: ";
41      cin >> r >> c;
42      Matrix M1( r , c );
43      cout<<"\nEnter the elements for the 1st Matrix:\n";
44      cin >> M1;
45      cout<<"\nEnter the dimension of the 2nd Matrix: ";
46      cin >> r >> c;
47      Matrix M2( r , c );
48      cout<<"\nEnter the elements for the 2nd Matrix:";
49      cin >> M2;
50      cout<<"\nThe 1st Matrix:\n";
51      cout << M1;
52      cout << "\nThe 2nd Matrix:\n";
53      cout << M2;
54  }
```

```
Enter the dimension of the 1st Matrix: 2  3

Enter the elements for the 1st Matrix:
11 22 33
44 55 66

Enter the dimension of the 2nd Matrix:  3 2

Enter the elements for the 2nd Matrix:
99 88
77 66
55 44

The 1st Matrix:
11      22      33
44      55      66

The 2nd Matrix:
99      88
77      66
55      44
```

Overloading special operators…

Overloading shorthand operator +=

```
 3  class Point
 4  {
 5          int x,y;
 6      public:
 7          Point()
 8          { x = y = 0; }
 9          Point( int a, int b )
10          { x = a;    y = b; }
11
12
13
14
15
16
17          void show_points();
18  };
```

```
25  int main()
26  {
27      Point P1(10,20), P2(11,22);
28
29      P1 += P2 ;
30
31      cout<<"\nP1=";  P1.show_points();
32      cout<<"\nP2=";  P2.show_points();
33  }
```

```
P1=( 21 , 42)
P2=( 11 , 22)
```

Overloading shorthand operator +=

```
3   class Point
4   {
5           int x,y;
6       public:
7           Point()
8           { x = y = 0; }
9           Point( int a, int b )
10          { x = a;    y = b; }
11
12          void operator += ( Point  P )
13          {
14              x = x + P.x;
15              y = y + P.y;
16          }
17          void show_points();
18  };
```

```
25  int main()
26  {
27      Point P1(10,20), P2(11,22);
28
29      P1 += P2 ;
30
31      cout<<"\nP1=";  P1.show_points();
32      cout<<"\nP2=";  P2.show_points();
33  }
```

```
P1=( 21 , 42)
P2=( 11 , 22)
```

Overloading function call operator ()

```
3   class Point
4   {
5       int x,y;
6   public:
7       Point()
8       { x = y = 0; }
9       Point( int a, int b )
10      { x = a;    y = b; }
11
12      Point  operator + ( Point P )
13      {
14          return Point(x + P.x,y + P.y);
15      }
16      Point operator ( ) (int a, int b)
17      {
18          x = a; y = b;
19          return *this;
20      }
21      void show_points();
22  };
```

```
32  int main()
33  {
34      Point P1(10,20), P2, P3, P4;
35
36      P2(100,200);
37
38      cout<<"\nP1=";  P1.show_points();
39      cout<<"\nP2=";  P2.show_points();
40
41      P3 = P1 + P2(20,25);
42      P4 = P2(11,21);
43      cout<<"\nP2=";  P2.show_points();
44      cout<<"\nP3=";  P3.show_points();
45      cout<<"\nP4=";  P4.show_points();
46  }
```

Void pointers

```
 3  int main()
 4  {
 5      int intvar = 10;
 6      float flovar = 12.5;
 7
 8      int* ptr_int;
 9      float* ptr_float;
10      void* ptrvoid;
11      ptr_int = &intvar;          //ok, int* to int*
12      // ptr_int = &flovar;     //error, float* to int*
13      // ptr_float = &intvar;   //error, int* to float*
14      ptr_float = &flovar;        //ok, float* to float*
15      ptrvoid = &intvar;          //ok, int* to void*
16      cout<<"content of ptrvoid: "<< *(int*)ptrvoid;
17      ptrvoid = &flovar;            //ok, float* to void*
18      cout<<"\ncontent of ptrvoid: "<< *(float*)ptrvoid;
19      return 0;
20  }
```

OUTPUT:

```
content of ptrvoid: 10
content of ptrvoid: 12.5
```

output

```
P1=( 10 , 20)
P2=( 100 , 200)
P2=( 11 , 21)
P3=( 30 , 45)
P4=( 11 , 21)
```

Function pointer

```
3   float avg(float a,float b)
4   {
5       return (a+b)/2;
6   }
7   int max(int a, int b)
8   {
9       return (a>b)?a:b;
10  }
11  int min(int a, int b)
12  {
13      return (a<b)?a:b;
14  }
```

```
15  int main()
16  {
17      int n1=10 ,n2=20, n3,n4;
18      float f1=10,f2=14,f3;
19      int (*f_ptr_int)( int, int );
20      float (*f_ptr_float)( float ,float );
21      f_ptr_int = max;
22      n3 = f_ptr_int(n1,n2);
23
24      cout<<"\n max(n1,n2)="<<n3;
25
26      f_ptr_int = min;
27      n4 = f_ptr_int(n1,n2);
28      cout<<"\n min(n1,n2)="<<n4;
29
30      f_ptr_float = avg;
31      f3 = f_ptr_float(f1,f2);
32      cout<<"\n avg(f1,f2)="<<f3;
33  }
```

```
max(n1,n2)=20
min(n1,n2)=10
avg(f1,f2)=12
```

```
3  void show(char s[])
4  {
5      cout<<s;
6  }
7  void some_function(void (*fp)(char[]), int x)
8  {
9      char msg[] = "msg to show function()";
10     fp(msg);
11     cout<<"\n x="<<x;
12 }
13 int main()
14 {
15     int a = 22;
16     void (*f_ptr)( char[] );
17     f_ptr = show;
18     some_function( f_ptr , a );
19 }
```

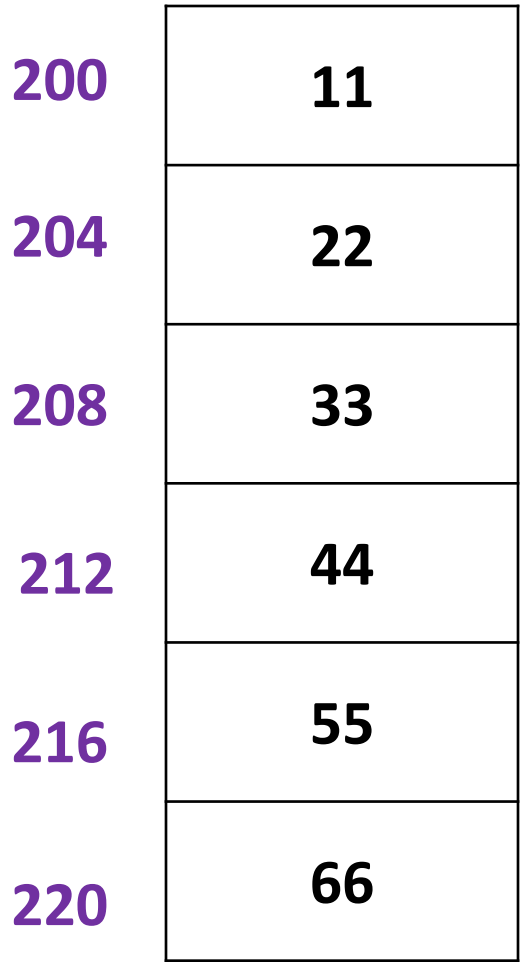
```
msg to show function()
 x=22
```

Pointer to pointer

```
 3  int main()
 4  {
 5      int a = 15;
 6      int *aptr , **bptr , ***cptr;
 7      aptr = &a;
 8      bptr = &aptr;
 9      cptr = &bptr;
10      cout<<"\n *aptr = "<< *aptr;
11      cout<<"\n **bptr = "<< **bptr;
12      cout<<"\n ***cptr = "<< ***cptr;
13  }
```

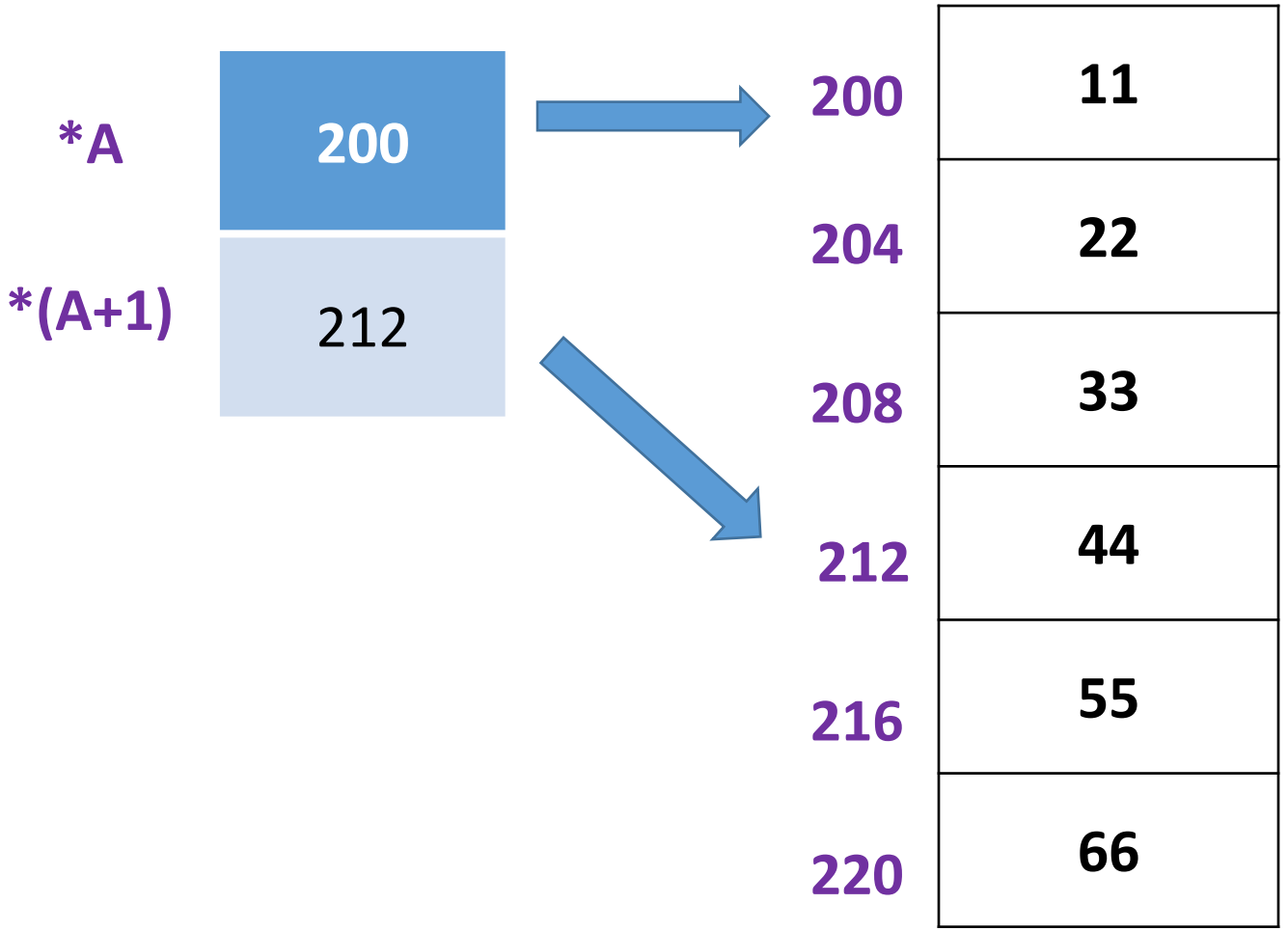
int A[2][3]

11	22	33
44	55	66



int A[2][3]

11	22	33
44	55	66



int A[2][3]

*(A+0)	11	22	33
*(A+1)	44	55	66

A[row_ind] [col_ind]

***(*(A + row_ind) + col_ind)**

*(*(A + 0) + 0)	11
*(*(A + 0) + 1)	22
*(*(A + 0) + 2)	33
*(*(A + 1) + 0)	44
*(*(A + 1) + 1)	55
*(*(A + 1) + 2)	66

```
 3  int main()
 4  {
 5      int  A[][3] = { {11, 22, 33} , {44, 55, 66} };
 6
 7      int row_size = 2, col_size = 3;
 8
 9      for(int i = 0 ; i < row_size ; i++ )
10      {
11          for( int j = 0 ; j < col_size ; j++ )
12
13              cout<< * ( *( A + i ) + j ) <<"\t";
14
15          cout<<endl;
16      }
17  }
```

Friend class

❖It is sometimes useful to allow a particular class to access private members of other class. For example a LinkedList class may be allowed to access private members of Node.

```
 6  class Node
 7  {
 8  private:
 9     int key;
10     Node *next;
11     /* Other members of Node Class */
12
13     friend class LinkedList; // Now class  LinkedList can
14                              // access private members of Node
15  };
```

COMMAND LINE ARGUMENTS

```
3   int main( int argc, char* argv[])
4   {
5       for( int i = 0; i < argc; i++ )
6           cout<<argv[i]<<"\n";
7   }
```

./a.out one two three 1 2 3

```
./a.out
one
two
three
1
2
3
```

Manipulators

- Manipulators are operators used in C++ for formatting output.

  — endl

  — setw

  — setfill

  — setbase

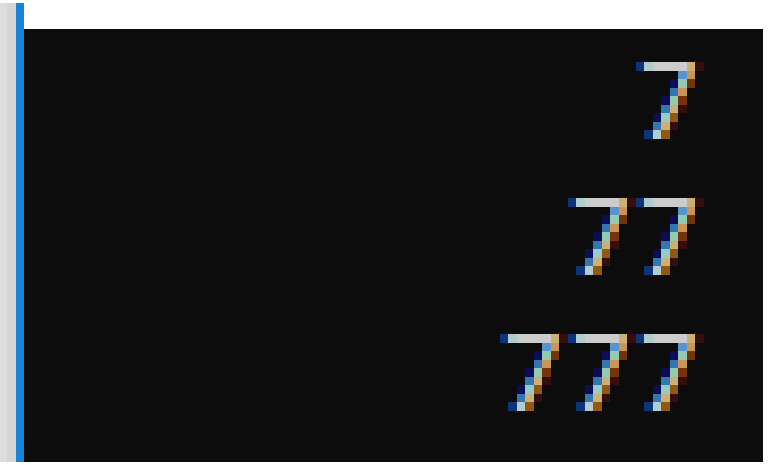
  — setprecision

  ……

- **setw**: manipulator sets the width of the field assigned for the output.

Syntax: Setw(n) , n –> Number of characters to be used as field width.

```
 4  int main()
 5  {
 6      int a=7, b=77, c=777;
 7      cout <<setw(10);
 8      cout <<a;
 9      cout <<"\n"<<setw(10)<<b<<"\n"<<setw(10)<<c;
10      return 0;
11  }
```



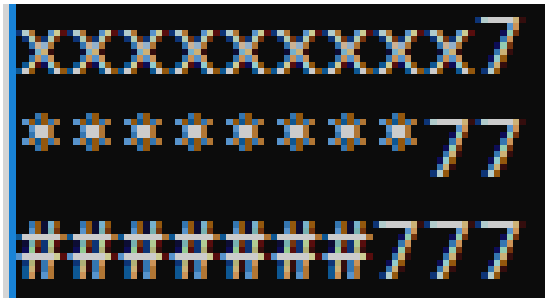
- **setfill** character is used to fill spaces when results have to be padded to the field width.

```
int a=7, b=77, c=777;

cout <<setfill ('x') << setw(10);
cout <<a;

cout <<"\n"<< setfill ('*') << setw(10) << b <<"\n";

cout<< setfill ('#' ) << setw(10) << c;
```



- **setprecision**

✓The setprecision Manipulator is used to set the number of digits printed to the right of the decimal point.

✓This may be used in two forms:

- ✓fixed
- ✓scientific

```
float x = 3.142857;
cout << fixed << setprecision(5) << x << endl;
cout << fixed << setprecision(3) << x << endl;
cout << scientific <<setprecision(3) <<x << endl;
```

```
3.14286
3.143
3.143e+000
```

- setbase (int base);

decimal : if base is 10
hexadcimal : if base is 16
octal : if base is 8
zero : if base is any other value.

```
ff
377
255
```

```
int x = 255;
// set base to hexadecimal
cout << setbase(16);

// displaying 255 in hexadecimal
cout << x <<endl;

// set base to Octal
cout << setbase(8);

// displaying 255 in Octal
cout << x << endl;

// displaying 255 in decimal
cout<< setbase(10);
cout<<x;
```