# Chapter 3

## Objective

- Assigning Objects
- Constructor And Destructors Functions
- Pointer To Object
- Enumeration Type as Constructor Parameter
- Dynamic Allocation of a Character String
- Initializing Array Of Object Using Constructor
- Pointer Arithmetic With Object
- Two-dimensional Array Of Objects
- Constructor Taking More Arguments
- Default Arguments
- Function Overloading
- Overloading An Inline Function
- Overloading The Constructor
- Passing Objects To Functions

# Unit 1

## Assigning Objects

- One object may be assigned to another object if both objects are of same type.
- Assignment makes a member to member copying of one object into another.
- Data becomes identical between the two objects.
- If data member is a pointer type the address within the data member becomes identical
- The two objects reside in a separate part of the memory.

```
1 #include <iostream>    // Example 3-1
2 using namespace std;
3
4 class CArea {
5  public:
6              void Assign(int h, int w) { m_nHeight = h; m_nWidth = w; }
7              void SetArea(){ m_nArea = m_nHeight*m_nWidth; m_ptrArea = &m_nArea; }
8              void Display();
9  private:
10             int m_nHeight;
11             int m_nWidth;
12             int m_nArea;
13             int *m_ptrArea;
14 };
15
```

# Unit 1

```
16 void CArea::Display()
17 {
18   cout << "Height:" << m_nHeight << " Width:" << m_nWidth << "\n";
19   cout << "The Area is: " << *m_ptrArea << "\n";
20 }
21
22   int main()
23   {
24      CArea Box1, Box2;
26      Box1.Assign(10,4);
27      Box1.SetArea();
28      Box2 = Box1;    // assign box1 to box2
29      Box1.Display();
30      cout << "Box2 value AFTER assignment\n";
31      Box2.Display();
33      Box1.Assign(20,5);
34      Box1.SetArea();
35      cout << "New value for Box1\n";
36      Box1.Display();
37      Box2.Display();
38   }
```

OUTPUT:

```
Height:10 Width:4
The Area is: 40
Box2 value AFTER assignment
Height:10 Width:4
The Area is: 40
New value for Box1
Height:20 Width:5
The Area is: 100
Height:10 Width:4
The Area is: 100
```

# Unit 2

## Default Arguments

•Formal parameters for a function can be given default arguments.

```
int correct(int x, int y=0, int z=0)  {   }
int illegal(int x=0, int y, int z)   {   }
```

•Only the rightmost trailing parameters can be unspecified when passing the parameter (calling the function).

```
1  #include <iostream>    // Example 3-10
2  using namespace std;
3
4  class CDate {
5     public:
6        void SetDate(int m,int d,int y);
7        void PrintDate();
8     private:
9        int m_nMonth;
10        int m_nDay;
11        int m_nYear;
12 };
13
```

```
14  void CDate::SetDate(int m=12, int d=25, int y=88)
15  {
16      m_nMonth = m;
17      m_nDay = d;
18      m_nYear = y;
19  }
20
21  void CDate::PrintDate()
22  {
23      cout << m_nMonth << "/" << m_nDay << "/" << m_nYear  << "\n";
24  }
25
```

```
26  int main()
27  {
28      CDate Birthday, Christmas, Payday;
29
30      Birthday.SetDate(9, 5, 65);    // default are overridden
31      Birthday.PrintDate();
32
33      Christmas.SetDate();    // all of the default are used
34      Christmas.PrintDate();
35
36      Payday.SetDate(1, 15);// part of argument are specified
37      Payday.PrintDate();
38  }
39
```

OUTPUT:
9/5/65
12/25/88
1/15/88

# Unit 3

## Constructor and Destructors Functions

Constructor Behavior

- Constructor is a member function whose name is same as the class name.
- Constructor is automatically called when creating an object on stack or using a new operator.
- Constructor can be placed in a public section or in a private section the class.
- Constructor supports most of the features available for standard functions.
- Constructor can not have a return statement.
- Constructors can not take arguments of its own class type pass by the value.
- Constructors can take a reference operator type arguments of its own class type.
- Default constructor has no arguments.

Destructor Behavior

- Destructor is a member function whose name is the same as the class name.
- Destructor is preceded by the character ~ (tilde)
- Destructor does not use any arguments and does not return any values.
- Destructor is automatically called when object goes out of scope.
- Destructors are not always needed.
- Destructor cannot be overloaded. (Compiler cannot distinguish same name)

# Unit 3

## Example of Constructor and Destructor

```
0 #include <iostream>    // Example 3-2
1
2 using namespace std;
3
4 class CTime {
5    public :
6     CTime();    // constructor
7     ~CTime() { cout << "Destructing: going out of scope...\n";}
8     void Show();
9     private:
10       int m_nHours;
11       int m_nMinutes;
12       int m_nSeconds;
13 };
14
```

```
15  CTime::CTime()
16  {
17      cout << "Inside constructor\n";
18      m_nHours = 12;
19      m_nMinutes = 30;
20      m_nSeconds = 45;
21  }
22
23  void CTime::Show()
24  {
25      cout << "The current time is: "
26      cout << m_nHours << ":";
27      cout << m_nMinutes << ":";
28      cout << m_nSeconds << " AM\n";
29  }
30
31
32  int main()
33  {
34      CTime Appointment;
35      Appointment.Show();
36  }
```

OUTPUT:
Inside constructor
The current time is: 12:30:45 AM
Destructing: going out of scope...

# Unit 4

## Pointer to Object

•Objects can be access using an arrow operator.

•Object pointer is declared just like a pointer to any other type of variable.

•Pointer arithmetic for an object is same as it is for any other data type.

```
1  #include <iostream>    // Example 3-3
2  using namespace std;
3  #define STACK_SIZE 10
5 class CStack {
6  private:
7               long books[STACK_SIZE];
8               int m_nSize;
10  public:
11              CStack() { m_nSize = -1; }
12              long top() { return(books[m_nSize]);}
13              long pop() { return (!Empty() ? books[m_nSize--] : -1);}
14              void push(long i) { if(!Full()) books[++m_nSize] = i;}
15              int Empty() { return (m_nSize < 0 ? 1 : 0); }
16              int Full() { return (m_nSize == STACK_SIZE ? 1 : 0); }
17              long bottom() { return books[0]; }
18 };
```

# Unit 4

## Pointer to Object

```
19 int main()
20 {
21          CStack *q = new CStack;
22
23          for (int i = 0; i < STACK_SIZE; i++) q->push(i+2);
24
25          cout << "Top of the stack book# is: " << q->top() << '\n';
26          for (int i = 0; i < STACK_SIZE; i++)
27          cout << q->pop() << "  ";
28
29          cout << "\nBottom of the stack book# is: " << q->bottom() << '\n';
30
31          delete q;
32 }
```

OUTPUT:

**Top of the stack book# is:: 11**

**11 10 9 8 7 6 5 4 3 2**

**Bottom of the stack book# is:: 2**

# Unit 5

## Enumeration Type as Constructor Parameter

```
1  #include <iostream>    // Example 3-4
2  using namespace std;
3
4  enum SignalColor {RED, ORANGE, GREEN};
5
6  class CTraffic {
7     public:
8        CTraffic(SignalColor eWhatcolor);
9        ~CTraffic();
10         SignalColor GetLight();
11     private:
12        SignalColor m_eLight;
13 };
14
```

# Unit 5

```
15  CTraffic::CTraffic(SignalColor eWhatColor)
16  {
17     cout << "Traffic Is Active\n";
18     m_eLight = eWhatColor;
19  }
20
21  CTraffic::~CTraffic()
22  {
23     cout << "Power Shortage disconnecting all traffic lights!!!\n";
24  }
25
26  SignalColor CTraffic::GetLight()
27  {
28     return(m_eLight);
29  }
30
```

# Unit 5

```
31  int main()
32  {
33     CTraffic SignalLight(GREEN);
34
35     switch (SignalLight.GetLight()) {
36        case RED:
37           cout << "RED Light Is ON\n";
38           break;
39        case ORANGE:
40           cout << "Orange Light Is ON\n";
41           break;
42        case GREEN:
43           cout << "Green Light Is ON\n";
44           break;
45     } // end switch
46  }
```

OUTPUT:
Traffic Is Active
Green Light Is ON
Power Shortage disconnecting all traffic lights!!!

# Unit 6

## Dynamic Allocation of a Character String

```
1  #include <iostream>    // Example 3-5
2  #include <string>
3
4  using namespace std;
5
6  class CString {
7     public:
8        CString(char *pcStr);
9        ~CString();
10         void Show();
11     private:
12        char *m_pcStr;
13        int m_nLen;
14 };
15
```

# Unit 6

```
18  int main()
19  {
20
21      CString s1("This is a test"), s2("I like C++");
22      s1.Show();
24      s2.Show();
25
26  }
27
28  CString::CString(char *pcStr)
29  {
30      m_nLen = strlen(pcStr);
31      m_pcStr = new char[m_nLen+1];
32      strcpy(m_pcStr, pcStr);
33  }
34
35  CString::~CString()
36  {
37      cout << "Freeing p\n";
38      if (m_pcStr) delete m_pcStr;
39  }
```

# Unit 6

```
41
42   void CString::Show()
43  {
44      cout << m_pcStr << " - length: " << m_nLen;
45      cout << "\n";
46  }
47
```

**OUTPUT:**

**This is a test - length: 14**
**I like C++ - length: 10**
**Freeing p**
**Freeing p**

# Unit 7

## Initializing Array Of Object Using Constructor

```
0  #include <iostream>    // Example 3-6
1
2  using namespace std;
3
4  class CFruit {
5     public:
6        CFruit(int n) { m_nCount = n; }
7        int Pick() { return m_nCount; }
8     private:
9        int m_nCount;
10 };
11
```

# Unit 7

```
12  int main()
13  {
14      int nIndex = 0;
15
16      // this will work if constructor has only one argument
17      CFruit anApples[4] = {10, 20, 30, 40};
18  //    OR
19      CFruit anApples[4]={
20          CFruit(10),
21          CFruit(20),
22          CFruit(30),
23          CFruit(40)
24      };
25
26      while (nIndex < 4) cout << anApples[nIndex++].Pick() << ' ';
27      cout << "\n";
28  }
```

OUTPUT:

**10 20 30 40**

# Unit 7

## Pointer Arithmetic With Object

```
1  #include <iostream>    // Example 3-7
2  using namespace std;
3
4  class CArea {
5     public:
6        CArea(int n, int m) { m_nHeight = n; m_nWidth = m; }
7        int Fetch_h()  { return m_nHeight; }
8        int Fetch_w()  { return m_nWidth; }
9     private:
10       int m_nHeight;
11       int m_nWidth;
12 };
13
```

# Unit 7

```
14  int main()
15  {
16      CArea aBox[4] = {
17          CArea(1, 2),
18          CArea(3, 4),
19          CArea(5, 6),
20          CArea(7, 8)
21      };
22
23      CArea *pBox;
24
25      pBox = aBox;    // get starting address of array
26
27      for (int i = 0; i < 4; i++) {
28          cout << pBox->Fetch_h() << ' ';
29          cout << pBox->Fetch_w() << "\n";
30          pBox++;    // advance to next object
31      }
32     cout << "\n";
33  }
```

OUTPUT:

```
1 2
3 4
5 6
7 8
```

# Unit 7

## Two-dimensional Array Of Objects

```
0  #include <iostream>    // Example 3-8
1
2  using namespace std;
3
4  class CFruit {
5     public:
6        CFruit(int n)  { m_nCount = n; }
7        int Pick() { return m_nCount; }
8     private:
9        int m_nCount;
10 };
11
```

# Unit 7

```
12  int main()
13  {
14      CFruit aoApples[4][2] = {   1, 2,
15                                  3, 4,
16                                  5, 6,
17                                  7, 8
18      };
19
20      for (int nRow = 0; nRow < 4; nRow++) {
21              for (int nColumn = 0; nColumn < 2; nColumn++)
22          cout << aoApples[nRow][nColumn].Pick() << ' ';
23       cout << "\n";
24
25      } // end for loop
26  }
```

OUTPUT:

```
1 2
3 4
5 6
7 8
```

# Unit 7

## Using Array of Objects With Constructor Taking More Arguments

```
1  #include <iostream>    // Example 3-9
2  using namespace std;
3
4  enum position {ROW=4, COLUMN=2};
5
6  class CArea {
7     public:
8        CArea(int n, int m) { m_nHeight = n; m_nWidth = m; }
9        int FetchHeight()  { return m_nHeight; }
10         int FetchWidth()  { return m_nWidth; }
11     private:
12        int m_nHeight;
13        int m_nWidth;
14 };
15
```

# Unit 7

```
16  int main()
17  {
18     CArea aBox[ROW][COLUMN] = {
19        CArea(1,2),   CArea(3,4),
20        CArea(5,6),   CArea(7,8),
21        CArea(9,10),  CArea(11,12),
22        CArea(13,14), CArea(15,16)
23     };
24
25     for (int nRow = 0; nRow < ROW; nRow++) {
26        for (int nColumn = 0; nColumn < COLUMN; nColumn++) {
27           cout << aBox[nRow][nColumn].FetchHeight() << ' ';
28           cout << aBox[nRow][nColumn].FetchWidth() << "\n";
29        } // end inner for loop
30     } // end for loop
31
32     cout << "\n";
33  }
```

OUTPUT:

```
1 2
3 4
5 6
7 8
9 10
11 12
13 14
15 16
```

# Unit 8

## Function Overloading

- Natural languages such as English have same spelling words if used in different context can change the meaning of the word; for example
- Overloading refers to using the same function name for multiple implementations.
- Compiler selects the function by matching the argument list.
- Apply a standard type conversion to the arguments then looks for a match.

```
1  #include <iostream>    // Example 3-11
2  using namespace std;
3
4  class CEmployee {
5     public:
6        double Average(double pay[], int size);
7        double Average(int hours[], int size);
8  };
9
```

# Unit 8

```
10  int main()
11  {
12
13      int hours[5] = {1, 2, 3, 4, 5};
14      double pay[5] = {1.1, 2.2, 3.3, 4.4, 5.5};
15      CEmployee PayCheck;
16
17      cout << PayCheck.Average(hours, 5);
18      cout << " int array average\n";
19      cout << PayCheck.Average(pay, 5);
20      cout << " double array average\n";
21  }
22
```

# Unit 8

```
23  double CEmployee::Average(int hours[], int size)
24  {
25     int sum = 0;
26     for (int index = 0; index < size; ++index) sum += hours[index];
27
28     return((double)sum/size);
29  }
30
31  double CEmployee::Average(double pay[], int size)
32  {
33    double sum = 0.0;
34    for (int index = 0; index < size; ++index) sum += pay[index];
35
36     return(sum/size);
37  }
```

OUTPUT:

**3 int array average**
**3.3 double array average**

# Unit 8

## Overloading An Inline Function

```
1  #include <iostream>    // Example 3-12
2  using namespace std;
3
4  // Overload abs() three ways
5  inline int abs(int n)
6  {
7     cout << "In integer abs() \n";
8     return( (n < 0) ? -n : n);
9  }
10
11  inline long abs(long n)
12  {
13     cout << "In long abs() \n";
14     return( (n < 0) ? -n : n);
15  }
16
17  inline double abs(double n);
18
```

# Unit 8

## Overloading An Inline Function

```
19  int main()
20  {
21     cout << "Absolute value of -10 Is ";
22     cout << abs(-10) << "\n";
23     cout << "Absolute value of -10L Is ";
24     cout << abs(-10L) << "\n";
25     cout << "Absolute value of -10.01 Is ";
26     cout << abs(-10.01) << "\n";
27  }
28
29  double abs(double n)
30  {
31     cout << "In double abs() \n";
32     return( (n < 0) ? -n : n);
33  }
34
```

OUTPUT:

```
In integer abs()
Absolute value of -10 Is 10
In long abs()
Absolute value of -10L Is 10
In double abs()
Absolute value of -10.01 Is 10.01
```

# Unit 8

## Overloading The Constructor

```
0  #include <iostream>    // Example 3-13
1  #include <cstring>      // used for strcpy function
2  #include <cstdlib>    // user for atoi function
3  using namespace std;
4  #define SIZE 11
5  class CView {
6     public:
7        CView() { memset(m_saOutString,'\0',SIZE); }
8        CView(char* pString) { strcpy(m_saOutString,pString); }
9        CView(int nNumber);
10       void Show() { cout << m_saOutString << '\n'; }
11    private:
12       char m_saOutString[SIZE];
13 };
14
```

# Unit 8

```
15  int main()
16  {
17      CView MyString("Count:");
18      CView MyNumber(7);
19      MyString.Show();
20      MyNumber.Show();
21  }
22
23  CView::CView(int nNumber)
24  {
25      char TmpStr[SIZE];
26
27      itoa(nNumber,TmpStr,10);
28      strcpy(m_saOutString,TmpStr);
29  }
30
```

OUTPUT:
Count:
7

# Unit 9

## Passing Objects to Functions

•Objects may be passed as function arguments.

•Declare formal parameter of function as a class type.

•By default all objects are passed by value to a function.

```
1  #include <iostream>    // Example 3-14
2  using namespace std;
3  class CFruit {
5     public:
6        CFruit(int n) { m_nCount = n; }
7        int Pick() { return m_nCount; }
8        void Set(int n) { m_nCount = n; }
9     private:
10        int m_nCount;
11 };
12
```

# Unit 9

```
13
14  // Non Member Function (Return square of m_nCount using arrow operator)
15  void SqrIt(CFruit *f)
16  {
17      f->Set(f->Pick() * f->Pick());
18  }
19
13  // Non member function (Return square of m_nCount using dot operator)
14  int SqrIt(CFruit f)
15  {
16      return(f.Pick() * f.Pick());
17  }
```

# Unit 9

```
20  int main()
21  {
22      CFruit Apples(10), Plums(2);
23
24      cout << "Apples count:" << Apples.Pick() << '\n ';
24      cout << "Plums count: " << Plums.Pick() << '\n ';
26      SqrIt(&Apples);
24      cout << "Apples count:" << Apples.Pick() << '\n ';

25      cout << SqrIt(Apples) << "\n";
26      cout << SqrIt(Plums) << "\n";
29  }
30
```

OUTPUT:

Apples count:10
Plums count:2
Apples count:100
10000
4