Chapter 2

Objective

- Using Namespace
- Rules for member function scope within a class.
- Understand why this Pointer is used.
- The Behavior of cin object.
- Using inline functions.
- Importance of enumeration types.
- Understanding new and delete operators.
- Dynamic allocation for Multi-dimensional Array.
- Creating singly link list using new operator.

Unit 1

Name Space

- A namespace is an abstract container or environment created to hold a logical grouping of unique identifiers
- An identifier defined in a namespace is associated with that particular namespace.
- The same identifier can be independently defined in multiple namespaces.
- Namespaces provide a mechanism for hiding local identifiers.
- A namespace is defined with a namespace block.
-

```
namespace first {
  int Number = 100;
}

namespace second {
   double Number = 3.1416;
}
```

- Within this block, identifiers can be used exactly as they are declared.
- Outside of this block, the namespace specifer must be prefixed with namespace name.

Name Space (continued)

```
#include <iostream>
int main()
{
  std::cout << first::Number << '\n';
  std::cout << second::Number << '\n';
}
```

if "using namespace first" construct is used than prefixed is not needed.

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

int main()
{
  cout << Number << '\n';
  cout << second::Number << '\n';
}
```

Unit 2

Member function Scope within the class

- Places each function within the scope of the class to which it belongs.
- Each member function enjoys special access privileges to other members of the class.
- No need to specify the class instance to access the class data members.
- Member functions do not take up any room inside the object.

```
0   #include <iostream>    // Example 2-1
1   using namespace std;
2
3   class CRectangle {
4      private:
5           int m_nWidth;
6           int m_nHeight;
7
8      public:
9           void SetSize(int w, int h);
10          int Area();
11   };
12
```

Member function Scope within the class (Continued)

```
13   void CRectangle::SetSize(int w, int h)
14   {
15       m_nWidth = w;
16       m_nHeight = h;
17   }
18
19   int CRectangle::Area()
20   {
21       return(m_nWidth * m_nHeight);
22}
24   int main()
25   {
26       CRectangle Rect1, Rect2;
27       Rect1.SetSize(10,20);
28       Rect2.SetSize(100,200);
29       cout << "The area of the first rectangle is ";
30       cout << Rect1.Area() << "\n";
31       cout << "The area of the second rectangle is ";
32       cout << Rect2.Area() << "\n";
33   }
```

Member function Scope within the class (Continued)

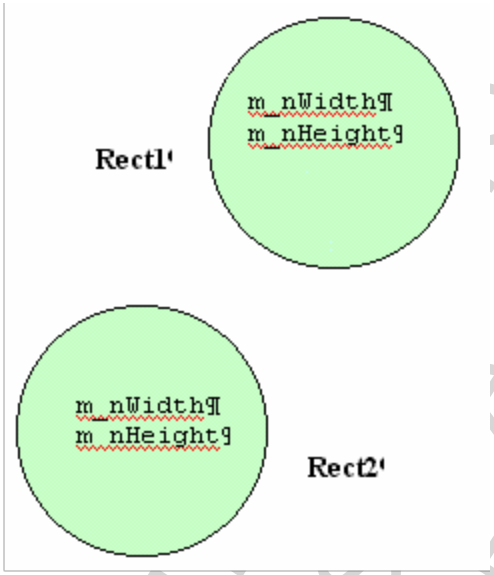

Figure 2.1

OUTPUT:

The area of the first rectangle is 200

The area of the second rectangle is 20000

The this Pointer

- **this** pointer is automatically passed when any member function is called.
- It is a pointer to the object that generates the call.
- **this** pointer **can not** be assigned any addresses, that is this = &Object // Error

```
0   #include <iostream>    // Example 2-2
1   #include <cstring>
2
3   using namespace std;
4
5   class CEmployee {
6       public:
7            void Init(const char *n, int id, int a);
8            void Display();
9       private:
10           char m_acName[20];
11           int m_nIdNum;
12           int m_nAge;
13   };
14
```

The this Pointer (Continued)

```
15
16   void CEmployee::Init(const char *n, int id, int a)
17   {    // access members through the this pointer
18        strcpy(this->m_acName,n);
19        this->m_nIdNum = id;
20        this->m_nAge = a;
21   }
22
23   void CEmployee::Display()
24   {
25        cout << this->m_acName;
26        cout << "ID: " << this->m_nIdNum;
27        cout << "  Age: " << this->m_nAge << "\n";
28   }
29
```

The this Pointer (Continued)

```
30   int main()
31   {
32       CEmployee WorkerData;
33       WorkerData.Init("Johnson",905,47);
34       WorkerData.Display();
35   }
36
```

```
OUTPUT:

JohnsonID: 905  Age: 47
```

Figure 2.2

Unit 3

Behavior of cin object

- cin object is of type istream class.
- It can call any public members from istream class using dot operator.

  - good() checks if the state of the stream is good for I/O operations.
  - good() returns true if none of the stream's error flags (eofbit, failbit, and badbit) are set.

  - clear() will reset the stream state back to a good state

  - ignore() will remove all remaining junk from the stream

Behavior of cin object (Continued)

```
0 #include <iostream>    // Example 2-3
1 #include <limits>
2 #include <cctype>
4 using namespace std;
6 int main()
7 {
8       int input = 0;
10      cout << "Enter the number: ";
12      while (cin >> input || !cin.eof()) {
20      if (cin.good()) {
21      if (input < 0) break;
22      else cout << "You Entered " << input << "\n";
23      } // end if
24      else if (!isdigit(input)) cout << "Not a number\n";
28      cin.clear();
32      cin.ignore(numeric_limits<streamsize>::max(),'\n');
33      cout << "Enter the number: ";
34      } // end while loop
35 }
```

Unit 4

Inline Functions

- To extend the power of macros, C++ provides inline functions.
- Macros in C language represent constants or a simple expression multiple times.
- Using the preprocessor to define macros has a drawback.
-
- ```
  #define SQ(X)    X*X
  call:    SQ(a+b)
  expands to:      a+b*a+b  multiplication will be done first.
  ```
-
- ```
  inline int SQ(int x)    {return(x*x);}
  ```
- Compiler treats **inline** function as **macro**.
- Code is inserted at the location of the call and appropriate variables are renamed.
- Inline function has the same scoping and argument passing semantics as standard functions.
- Compiler may choose to ignore inline keyword.
- Very short function should be declared as inline function.
-
- Two ways to declare inline function.
- Start function declaration using inline keyword.
- Include the entire function body inside the class template.

Inline Functions (Continued)

```
0   #include <iostream>    // Example 2-5
1   using namespace std;
2
3 inline double Round(double fAmount)
4 {
5       long lCents = long(100.0 * fAmount + 0.5);
6       return double(lCents) /100.0;
7 }
8
9       class CInvoice {
10      public:
11      void GetData();
12      void Display();
13      private:
14      double m_fPrice;
15      int m_nInvoiceNum;
16      };
```

Inline Functions (Continued)

```
17 int main()
18 {
19       CInvoice Items;
20       Items.GetData();
21       Items.Display();
22 }
23 void CInvoice::GetData()       // Get input data from user
24 {
25       cout << "Enter Invoice Number:";
26       cin >> m_nInvoiceNum;
27
28       cout << "Regular price: ";
29       cin >> m_fPrice;
30 }
```

Inline Functions (Continued)

```
31 void CInvoice::Display()       // Print results
32 {
33       cout << "Invoice Number = " << m_nInvoiceNum << '\n';
34       cout << "Regular Price = $" << Round(m_fPrice) << '\n';
35 }
36
```

INPUT:

```
Enter Invoice Number: 1024
Regular price:53.8573
```

OUTPUT:

```
Invoice Number = 1024
Regular Price = $53.86
```

Unit 5

Enumeration Types

- Enumeration is a user defined data type and it makes programs more readable.
- The enumeration type enables the **naming** of integer values defined in a finite set.
- Values are specified for the names that are used for identifying the enumeration constants.

```
enum travel {SEA, AIR, ROAD};
```

-
- Integer values can be assigned to the corresponding names in the enumeration set.

```
enum travel {SEA=1, AIR=2, ROAD=3};
```

- By default each enumeration constant is one greater than the previous constant, unless the value is specified.

```
enum color {RED = 5, YELLOW, GREEN = 13, BLUE};
```

- ASCII values of characters can be assigned to corresponding names in the enumeration set.

```
enum commands {PARK='p', RETRIVE='r', DISPLAY='d', EXIT='e'};
```

- Use tag name when declaring variables of type enumeration.
- Enumeration type is converted to type int before any arithmetic operations are performed.

Enumeration Types (Continue)

```
1   #include <iostream>    // Example 2-6
2   using namespace std;
3   enum Travel {SEA, AIR, ROAD};
4   int main()
5   {
6       Travel Journey = SEA;
7       Journey = (Travel)5;    // error!! meaningless value to journey
8       Journey = (Travel)2;    // journey gets the ROAD value
9      switch (Journey) {
10       case SEA:
11         cout << "traveling by sea\n";
12         break;
13       case AIR:
14         cout << "traveling by air\n";
15         break;
16       case ROAD:
17         cout << "traveling by road\n";
18         break;
19       default:
20         cerr << "Invalid Entry\n";
21         break;
22      }
23      int World = Journey; // OR World = int(Journey);
24  }  // end switch
25 }  // End main function
```

Arrays and Objects

- Array of objects is declared same way as of any other variable.

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

Arrays and Objects (Continue)

```
13   int main()
14   {
15       CFruit Apples[10];
16
17       for (int nIndex = 0;
18           nIndex < sizeof(Apples)/sizeof(Apples[0]); nIndex++)
19           Apples[nIndex].Assign(nIndex + 3);
20
21       for (int nIndex = 0;
22           nIndex < sizeof(Apples)/sizeof(Apples[0]); nIndex++)
23           cout << Apples[nIndex].Pick() << "  ";
24
25       cout << "\n";
26   }
```

OUTPUT:

```
3  4  5  6  7  8  9  10  11  12
```

Unit 6

Memory Management

- **new** and **delete** Operators are used for memory management.
- new and delete operator create and destroy data object at programmers will.
- Data object created by new operator stays in scope throughout the program.
- By using delete operator data object goes out of scope (destroyed)

```
0  #include <iostream>    // Example 2-8
1
2  using namespace std;
3
4  int main()
5  {
6  // Allocate memory for single integer type
7  int *ptrNum = new int;
8  *ptrNum = 100;
9
10 cout << "The Number is: " << *ptrNum << '\n';
11 delete ptrNum;
12
```

Memory Management (Continue)

```
13      // Allocate memory for single character type
14      char *ptrChar = new char;
16      *ptrChar = 'S';
17      cout << "The Character is: " << *ptrChar << '\n';
18      delete ptrChar;
19
20      // Allocate memory for array of integers
21      int *arrayNum = new int [10];
22      for (int i = 0; i < 10; i++) arrayNum[i] = i+1;
23      for (int i = 0; i < 10; i++) cout << arrayNum[i] << " ";
24      delete [] arrayNum;
25  }  // end of main
```

OUTPUT:

```
The Number is: 100
The Character is: S
1 2 3 4 5 6 7 8 9 10
```

Dynamic Allocation of Multi-dimensional Array

```
0   #include <iostream>    // Example 2-9
2   using namespace std;
4   int main()
5   {
7        int x, y, nNumber=10;
8        int **pnMatrix=0;
9        pnMatrix = new int*[nNumber];
11       for (x=0; x<nNumber; x++)
12          pnMatrix[x] = new int[nNumber];
13
14       for (x=0; x<nNumber; x++)
15          for (y=0; y < nNumber; y++)
16             *(pnMatrix[x] + y) = y;
17
18       for (x=0; x<nNumber; x++) {
19          for (y=0; y < nNumber; y++)
20              cout << pnMatrix[x][y];
21
22          cout << "\n";
23       } // end for loop
```

Dynamic Allocation of Multi-dimensional Array (Continue)

```
24
25        for (x=0; x<nNumber; x++)
26            delete [] pnMatrix[x];
27
28        delete [] pnMatrix;
29    }  // End of main
```

OUTPUT:

Dynamic Allocation of an Object

- Use new operator to allocate memory for an object
- Use delete operator to remove an object memory

```
0   #include <iostream>    // Example 2-10
1   #include <cstring>    // for memset function
2
3    using namespace std;
4
5    class CEmployee {
6  public:
7  void Init(int size);
8  void Set(const char *n, int id, int a);
9  void Display();
10 private:
11 char *m_pcName;
12 int m_nIdNum;
13 int m_nAge;
14  };
```

Dynamic Allocation of an Object (Continue)

```
15   void CEmployee::Init(int size)
16  {
17       m_pcName = new char [size];
18       memset(m_pcName, '\0', size+1);
19  }

20  void CEmployee::Set(const char *n, int id, int a)
21  {    // access members through the this pointer
22           strcpy(m_pcName,n);
23           m_nIdNum = id;
24           m_nAge = a;
25  }

27   void CEmployee::Display()
28   {
29           cout << "Name: " << m_pcName;
30           cout << "  ID: " << m_nIdNum;
31           cout << "  Age: " << m_nAge << "\n";
32           if (m_pcName) delete [] m_pcName;
33   }
```

Dynamic Allocation of an Object (Continue)

```
34   int main()
35   {
36       CEmployee *ptrWorkerData = new CEmployee;
37
38       ptrWorkerData->Init(20);
39       ptrWorkerData->Set("Johnson",905,47);
40       ptrWorkerData->Display();
41
42       delete ptrWorkerData;
43   }
```

OUTPUT:

```
Name: Johnson  ID: 905  Age:  47
```

Unit 7

Simple Link List

```
1   #include <iostream>    // Example 2-11
2
3   using namespace std;
4   class CNode {
5       public:
6            int m_nData;
7            CNode *m_pLink;
8   };

9   int main()
10   {
11       CNode *pHeadPtr, *pCurrPtr, *pTailPtr, *pDeleteThisNode;
12
13       pCurrPtr = new CNode;
14       pTailPtr = pHeadPtr = pCurrPtr;
15       pCurrPtr->m_nData = 1;
16       pCurrPtr->m_pLink = 0;
17
```

Simple Link List (Continue)

```
18        pTailPtr->m_pLink = new CNode;
19        pCurrPtr = pTailPtr->m_pLink;
20        pCurrPtr->m_nData = 2;
21        pCurrPtr->m_pLink = 0;
22        pTailPtr = pCurrPtr;
23
24        pTailPtr->m_pLink = new CNode;
25        pCurrPtr = pTailPtr->m_pLink;
26        pCurrPtr->m_nData = 3;
27        pCurrPtr->m_pLink = 0;
28        pTailPtr = pCurrPtr;
29
30        pCurrPtr = pHeadPtr;
31        while (pCurrPtr != 0) {
32          pDeleteThisNode = pCurrPtr;
33          cout << pCurrPtr->m_nData;
34          pCurrPtr = pCurrPtr->m_pLink;
35          delete pDeleteThisNode;
36        }
37    }  // End main function
```

Creating Link List Using for Loop

```
0   #include <iostream>    // Example 2-12
1   #include <cstring>
2
3   using namespace std;
4
5   // class definition to store employee information
8   class CEmployee {
9       public:
10            void Create(int TotalEmp);
11            void GetEmpData(CEmployee *pFreshNode);
12            void ComputeWage(CEmployee *pCurNode);
13            void Remove(CEmployee *pHeadNode);
14       private:
15            char m_aName[30];
16            unsigned int m_nAge;
17            unsigned int m_nSalary;
18            CEmployee *m_pLink;
19   };
20
```

Creating Link List Using for Loop (Continue)

```
21   void CEmployee::Remove(CEmployee *pHeadNode)
22   {
23       CEmployee *pCurr, *pDeleteThisNode;
24
25       pCurr = pHeadNode;
26       while (pCurr != 0) {
27       pDeleteThisNode = pCurr;
28       cout << "\nDeleting: \n" << pCurr->m_aName;
29       pCurr = pCurr->m_pLink;
30       delete pDeleteThisNode;
31       }
32   }
33
```

Creating Link List Using for Loop (Continue)

```
34   void CEmployee::GetEmpData(CEmployee *pFreshNode)
35   {
36       cout << "Enter the employee's name: ";
37       cin >> pFreshNode->m_aName;
38       cout << "Enter the employee's age: ";
39       cin >> pFreshNode->m_nAge;
40       cout << "Enter the employee's salary: ";
41       cin >> pFreshNode->m_nSalary;
42       cout << "\n";
43   }
44
```

Creating Link List Using for Loop (Continue)

```
45    void CEmployee::Create(int nTotalEmp)
46   {
47       CEmployee *pFreshNode, *pHeadNode, *pCurNode;
48
49       for (int i=0; i < nTotalEmp; i++) {
50          pFreshNode = new CEmployee;
51          GetEmpData(pFreshNode);
52          pFreshNode->m_pLink = 0;
53          if (i == 0) {
54            pHeadNode = pFreshNode;
55            pCurNode = pFreshNode;
56          }
57          else {
58             pCurNode->m_pLink = pFreshNode;
59             pCurNode = pFreshNode;
60          }
61       } // end for loop
62       pCurNode = pHeadNode;
63       ComputeWage(pCurNode);
64       Remove(pHeadNode);
65   }
```

Creating Link List Using for Loop (Continue)

```
67   void CEmployee::ComputeWage(CEmployee *pCurNode)
68   {
69       int nNumHours;  float fWeekPay; static char caSearchName[31];
70       cout << "\nEnter employee's Name to compute pay: ";
71       cin >> caSearchName;
72       while (pCurNode != 0) {
73           if (strcmp(caSearchName,pCurNode->m_aName) == 0) {
74              cout << "How many hours did ";
75              cout << pCurNode->m_aName << " work this week? ";
76              cin >> nNumHours;
77              fWeekPay = (pCurNode->m_nSalary / 2080) * nNumHours;
78              cout << "Annual salary is: ";
79              cout << pCurNode->m_nSalary << "\n";
80              cout << pCurNode->m_aName << "'s ";
81              cout << " pay this week is: " << fWeekPay;
82              break;
83            } // end if
84            else  pCurNode = pCurNode->m_pLink;
85       }  // end while
86 }
```

Creating Link List Using for Loop (Continue)

```
90   int main()
91   {
92        int nNumEmp;
93        cout << "Enter the number of employees to process: ";
94        cin >> nNumEmp;
95
96        CEmployee Worker;
97        Worker.Create(nNumEmp);
98    }
```

OUTPUT:

Enter the number of employees to process: 1
Enter the employee's name: Bob
Enter the employee's age: 35
Enter the employee's salary: 60000

Enter employee's Name to compute pay: Bob
How many hours did Bob work this week? 40
Annual salary is: 60000
Bob's pay this week is: 1120
Deleting:
Bob