



Module 17

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

friend Function

Matrix-Vector
Multiplication

Linked List

friend Class

Linked List

Iterator

Properties

Comparison

Module Summary

Module 17: Programming in C++

friend Functions and friend Class

Instructors: Abir Das and Sourangshu Bhattacharya

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

{*abir, sourangshu*}@cse.iitkgp.ac.in

Slides taken from NPTEL course on Programming in Modern C++

by **Prof. Partha Pratim Das**



Module Objectives

Module 17

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives & Outlines

friend Function

Matrix-Vector
Multiplication

Linked List

friend Class

Linked List

Iterator

Properties

Comparison

Module Summary

- Understand `friend` function and class



Module Outline

Module 17

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives & Outlines

friend Function

Matrix-Vector
Multiplication
Linked List

friend Class

Linked List
Iterator

Properties

Comparison

Module Summary

- 1 friend Function
 - Matrix-Vector Multiplication
 - Linked List
- 2 friend Class
 - Linked List
 - Iterator
- 3 Properties of friend
- 4 Comparison
- 5 Module Summary



Program 17.01: friend function: Basic Notion

Module 17

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

friend Function

Matrix-Vector
Multiplication

Linked List

friend Class

Linked List

Iterator

Properties

Comparison

Module Summary

Ordinary function

```
#include<iostream>
using namespace std;
class MyClass { int data_;
public:
    MyClass(int i) : data_(i) { }
};
void display(const MyClass& a) { // gbl. func.
    cout << "data = " << a.data_; // Error 1
}
int main() {
    MyClass obj(10);

    display(obj);
}
```

- `display()` is a non-member function
- **Error 1:** 'MyClass::data_' : cannot access private member declared in class 'MyClass'

friend function

```
#include<iostream>
using namespace std;
class MyClass { int data_;
public:
    MyClass(int i) : data_(i) { }

    friend void display(const MyClass& a);
};
void display(const MyClass& a) { // global function
    cout << "data = " << a.data_; // Okay
}
int main() {
    MyClass obj(10);

    display(obj);
}
```

- `display()` is a non-member function; but friend to class MyClass
- Able to access `data_` even though it is private in class MyClass
- **Output:** data = 10



friend function

Module 17

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

friend Function

Matrix-Vector
Multiplication

Linked List

friend Class

Linked List

Iterator

Properties

Comparison

Module Summary

- A **friend** function of a class
 - has access to the **private** and **protected** members of the class (*breaks the encapsulation*) in addition to **public** members
 - must have its *prototype included within the scope of the class* prefixed with the keyword **friend**
 - *does not have its name qualified* with the class scope
 - *is not called with an invoking object* of the class
 - *can be declared friend* in *more than one classes*
- A **friend** function can be a
 - *global function*
 - a *member function* of a class
 - a *function template*



Program 17.02: Multiply a Matrix with a Vector

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

class Matrix; // Forward declaration

class Vector { int e_[3]; int n_; public:
    Vector(int n) : n_(n) {
        for (int i = 0; i < n_; ++i) // Arbitrary
            e_[i] = i + 1;           // init.
    }
    void Clear() { // Set a zero vector
        for(int i = 0; i < n_; ++i)
            e_[i] = 0;
    }
    void Show() { // Show the vector
        for(int i = 0; i < n_; ++i)
            cout << e_[i] << " ";
        cout << endl << endl;
    }
    friend Vector Prod(Matrix *pM, Vector *pV);
};
```

```
class Matrix { int e_[3][3]; int m_, n_; public:
    Matrix(int m, int n) : m_(m), n_(n) { // Arbitrary
        for(int i = 0; i < m_; ++i)       // init.
            for(int j = 0; j < n_; ++j) e_[i][j] = i + j;
    }
    void Show() { // Show the matrix
        for (int i = 0; i < m_; ++i) {
            for (int j = 0; j < n_; ++j)
                cout << e_[i][j] << " ";
            cout << endl;
        } cout << endl;
    }
    friend Vector Prod(Matrix *pM, Vector *pV);
};

Vector Prod(Matrix *pM, Vector *pV) {
    Vector v(pM->m_); v.Clear();
    for(int i = 0; i < pM->m_; i++)
        for(int j = 0; j < pM->n_; j++)
            v.e_[i] += pM->e_[i][j] * pV->e_[j];
    return v;
}
```

- `Vector Prod(Matrix*, Vector*);` is a **global function**
- `Vector Prod(Matrix*, Vector*);` is **friend** of class `Vector` as well as class `Matrix`
- This function accesses the **private** data members of both these classes



Program 17.02: Multiply a Matrix with a Vector

Module 17

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

friend Function

Matrix-Vector
Multiplication

Linked List

friend Class

Linked List

Iterator

Properties

Comparison

Module Summary

```
int main() {  
    Matrix M(2, 3);  
    Vector V(3);  
  
    Vector PV = Prod(&M, &V);  
  
    M.Show();  
    V.Show();  
    PV.Show();  
  
    return 0;  
}
```

Output:

```
0 1 2 // Matrix M  
1 2 3  
  
1 2 3 // Vector V  
  
8 14 // Product Vector PV
```

- `Vector Prod(Matrix*, Vector*);` is a **global function**
- `Vector Prod(Matrix*, Vector*);` is **friend** of class `Vector` as well as class `Matrix`
- This function accesses the **private** data members of both these classes



Program 17.03: Linked List

Module 17

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

friend Function

Matrix-Vector
Multiplication

Linked List

friend Class

Linked List

Iterator

Properties

Comparison

Module Summary

```

#include <iostream>
using namespace std;

class Node;    // Forward declaration
class List {
    Node *head; // Head of the list
    Node *tail; // Tail of the list
public:
    List(Node *h = 0): head(h), tail(h) { }
    void display();
    void append(Node *p);
};

class Node {
    int info; // Data of the node
    Node *next; // Ptr. to next node
public:
    Node(int i): info(i), next(0) { }
    friend void List::display();
    friend void List::append(Node *);
};

void List::display() { // friend of Node
    Node *ptr = head;
    while (ptr) { cout << ptr->info << " ";
        ptr = ptr->next;
    }
}

void List::append(Node *p) { // friend of Node
    if (!head) head = tail = p;
    else {
        tail->next = p;
        tail = tail->next;
    }
}

int main() { List l; // Init. null list
    Node n1(1), n2(2), n3(3); // Few nodes
    l.append(&n1); // Add nodes to list
    l.append(&n2);
    l.append(&n3);
    l.display(); // Show list
}

```

- List is built on Node. Hence List needs to know the internals of Node
- void List::append(Node *); needs the internals of Node – hence friend member function is used
- void List::display(); needs the internals of Node – hence friend member function is used
- We can do better with friend classes



friend class

Module 17

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

friend Function

Matrix-Vector
Multiplication

Linked List

friend Class

Linked List

Iterator

Properties

Comparison

Module Summary

- A **friend** class of a class
 - has access to the **private** and **protected** members of the class (*breaks the encapsulation*) in addition to **public** members
 - *does not have its name qualified* with the class scope (*not a nested class*)
 - *can be declared friend* in *more than one classes*
- A **friend** class can be a
 - **class**
 - **class template**



Program 17.04: Linked List

Module 17

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

friend Function

Matrix-Vector
Multiplication

Linked List

friend Class

Linked List

Iterator

Properties

Comparison

Module Summary

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

class Node; // Forward declaration
class List {
    Node *head; // Head of the list
    Node *tail; // Tail of the list
public:
    List(Node *h = 0): head(h), tail(h) { }
    void display();
    void append(Node *p);
};

class Node {
    int info; // Data of the node
    Node *next; // Ptr to next node
public:
    Node(int i): info(i), next(0) { }
    // friend void List::display();
    // friend void List::append(Node *);
    friend class List;
};

void List::display() {
    Node *ptr = head;
    while (ptr) { cout << ptr->info << " ";
        ptr = ptr->next;
    }
}

void List::append(Node *p) {
    if (!head) head = tail = p;
    else {
        tail->next = p;
        tail = tail->next;
    }
}

int main() { List l; // Init null list
    Node n1(1), n2(2), n3(3); // Few nodes
    l.append(&n1); // Add nodes to list
    l.append(&n2);
    l.append(&n3);

    l.display(); // Show list
}
```

- List class is now a friend of Node class. Hence it has full visibility into the internals of Node
- When multiple member functions need to be friends, it is better to use friend class



Program 17.05: Linked List with Iterator

Module 17

Instructors: Abir Das and Sourangshu Bhattacharya

Objectives & Outlines

friend Function

Matrix-Vector Multiplication

Linked List

friend Class

Linked List

Iterator

Properties

Comparison

Module Summary

```

#include <iostream>
using namespace std;
class Node; class List; // Forward declarations
class Iterator { Node *node; // Current Node
                List *list; // Current List
public: Iterator() : node(0), list(0) { }
        void begin(List *); // Init
        bool end(); // Check end
        void next(); // Go to next
        int data(); // Get node data
};
class List { Node *head, *tail; public:
        List(Node *h=0): head(h), tail(h) { }
        void append(Node *p);
        friend class Iterator;
};
class Node { int info; Node *next; public:
        Node(int i) : info(i), next(0) { }
        friend class List;
        friend class Iterator;
};

// Iterator methods
void Iterator::begin(List *l) {
    list = l; node = l->head; // Set list & Init
}
bool Iterator::end() { return node == 0; }
void Iterator::next() { node = node->next; }
int Iterator::data() { return node->info; }

void List::append(Node *p) {
    if (!head) head = tail = p;
    else { tail->next = p; tail = tail->next; }
}
int main() { List l;
            Node n1(1), n2(2), n3(3);
            l.append(&n1); l.append(&n2); l.append(&n3);

            Iterator i;
            for(i.begin(&l); !i.end(); i.next()) {
                cout << i.data() << " "; // Iteration Loop
            }
}

```

- An Iterator now traverses over the elements of the List
- void List::display() is dropped from List and can be written in main()
- List class is a friend of Node class
- Iterator class is a friend of List and Node classes



Properties of friend

Module 17

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

friend Function

Matrix-Vector
Multiplication

Linked List

friend Class

Linked List

Iterator

Properties

Comparison

Module Summary

- **friendship** is neither *commutative* nor *transitive*
 - A is a **friend** of B *does not imply* that B is a **friend** of A
 - A is a **friend** of B and B is a **friend** of C *does not imply* that A is a **friend** of C
- **Visibility and Encapsulation**
 - **public**: a declaration that is accessible to all
 - **protected**: a declaration that is accessible only to the class itself and its subclasses
 - **private**: a declaration that is accessible only to the class itself
 - **friend**: a declaration that is accessible only to **friend**'s of a class. **friends** tend to *break data hiding* and **must be used judiciously**. Like:
 - ▷ A function needs to access the internals of two (or more) independent classes (Matrix-Vector Multiplication)
 - ▷ A class is built on top of another (List-Node Access, List Iterator)
 - ▷ Certain situations of operator overloading (like streaming operators)



Comparison of friend vis-a-vis Member Functions

Module 17

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

friend Function

Matrix-Vector
Multiplication

Linked List

friend Class

Linked List

Iterator

Properties

Comparison

Module Summary

friend Functions

- Declared using the keyword **friend**
- Declared in *one or more classes*
- *Not a part* of the class, *not defined* in the **namespace** of the classes
- Has *access to all* **private**, **public**, and **protected** members of classes

- May be *global* or *member function* of some other class
- Called with an *object* (non-**static** member), an *object* / a *class* (**static** member), or as a *global* function
- Does not have **this** pointer (of the class it accesses). Needs the *pointer to the object*
- Breaks *encapsulation*

static & Non-static Member Functions

- Declared in **private**, **public**, or **protected** specifier
- Declared *only in scope of a particular class*
- *Part* of the class definition, *defined* in the **namespace** of the class
- Has *access to all* **private**, **public**, and **protected** members of its class, if non-**static**
- Has *access to only* **private**, **public**, and **protected** **static** members of its class, if **static**
- *Member function* of the class
- Called with an *object* (non-**static** member) or an *object* / a *class* (**static** member) of the defining class
- Has **this** pointer of the defining class, if a Non-**static** and no **this** pointer if **static**
- Ensures *encapsulation*



Module Summary

Module 17

Instructors: Abir
Das and
Sourangshu
Bhattacharya

Objectives &
Outlines

friend Function

Matrix-Vector
Multiplication

Linked List

friend Class

Linked List

Iterator

Properties

Comparison

Module Summary

- Introduced the notion of `friend` function
- Introduced the notion of `friend` class
- Studied the use of `friend` function and `friend` class with examples
- `friend` introduces visibility hole by breaking encapsulation – should be used with care